Wireless Interconnection between Two FPGAs for FPGA Prototyping Applications

Made By

Ahmed Hani Mohamed Abdullah El Gohary
Sayed Shaban Sayed Mahmoud
Omar Mahmoud Yehia Mohamed
Mohamed Ahmed Thabet Zaki
Mohamed Mohamed Saad Mahmoud Ali
Mohannad Khaled Serag El Deen Asar

Under supervision of

Dr. Hassan Mostafa - CUFE
Dr. Eman El Mandouh – Mentor Graphics

A graduation project thesis submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Science

Cairo, Egypt
Department of Electronics and Communications Engineering

July 2018

Table of Contents

List of Figures

Acknowledgement

First of all, we thank God Almighty, who has granted us this opportunity and helped us reach this far and achieve this progress.

We'd like to thank our supervisors from the faculty of engineering and from Mentor Graphics. We'd like to thank Dr. Hassan Mostafa, Dr. Eman El-Mandouh, Eng. Ahmed Ismail, Eng. Sherif Saad, and all the members of the FPGA Prototyping team at Mentor Graphics.

We'd also like to thank those who helped us throughout the year; we thank Dr. Karim Osama, Dr. Mohamed Khairy, Eng. Mohamed Adel, Eng. Abdelrahman Mohsen, and all those who helped us from the staff of the Faculty of Engineering, Cairo University.

And finally, we'd like to thank our families and beloved friends, who have supported us and helped us reach this far.

Thank you, and may this help us all achieve our dreams and ambitions.

1. Introduction

1.1 FPGA Prototyping

FPGA prototyping is the method to implement system-on-chip (SoC) and ASIC designs on FPGAs for hardware verification and validation. Every ASIC design needs verification for all the possible test cases to ensure that no errors will appear after the fabrication, so the design is tested on the FPGA and its behavior will be closest to the fabricated IC, so it'll catch most hardware errors.

As complexity of different designs increases, the overall area increases and a single FPGA isn't enough to contain the whole design, so the solution is to either increase the area of the FPGA or to partition the design into more than one FPGA. The two solutions are discussed as follows.

The Emulator is a hardware that can be described as a huge FPGA that's large enough to contain any design even the most complex of them. The disadvantage of this device is its very slow speed, meaning that if a design operates at 1 GHz clock frequency after fabrication, it may not operate at a clock faster than 1 MHz on an emulator, which means the design won't work as expected, so the testing won't be accurate.

FPGAs on the other hand can operate at faster frequencies due to its lower area and complexity relative to the Emulator, so for the design that operates at 1 GHz frequency after fabrication, it can operate at around 20 MHz clock frequency on an FPGA. If we increase its area the frequency will decrease (An emulator is a huge FPGA) so we'll face the same problem of limited clock frequency, so the solution for this problem is FPGA partitioning.

FPGA partitioning means that if the design is too large to be tested on a single FPGA, it is partitioned into 2 sub-designs each one is on an FPGA, and the 2 FPGAs are connected by wires. If the design's area is even larger, it can be partitioned into more than 2 FPGAs and connect all of them by wires. For example, consider a design that includes a processor, a memory, and a DMA, and each of these blocks' area is large enough to fill an FPGA, then the design will be partitioned such that each one of them will take one FPGA, and they'll be connected together to transfer the required signals as if they're on a single FPGA.

1.2 Critical Issues

Unfortunately, FPGA Partitioning has two main problems which deeply affect the performance. These two problems are as follows:

1.2.1 Where the partition is done

Assuming the design is to be divided into 2 partitions, choosing the path that'll separate the 2 partitions isn't easy, as if this path is the critical path, then the design's maximum frequency will decrease as the connection between two FPAGs will add significant delay to the maximum delay in the design which increases the maximum delay and decreases the maximum clock frequency, thus the line of separation should contain the fastest paths only.

1.2.2 Limited number of pins

The pins of the FPGAs may not be enough to connect the two partitions. For example, assuming the design is divided into two partitions, if each FPGA has 1000 I/O pins and 800 of these pins are connected to inputs and outputs of the main design, then only 200 pins are available for the signals connecting the two parts. If the design has 400 signals that need to connect the 2 parts, then the pins are not enough. One of the solutions to this issue is to insert a 2-to-1 Mux that'll send the first 200 signals in the first half of the clock cycle, and the other 200 signals in the other half, but this means the signals would need to have a delay that's less than half of the max delay. As that's mostly not the case, the overall clock frequency of the design is decreased to allow the signals to be transferred successfully.

1.2.3 Multiple-FPGAs Communication

For a design that's partitioned on more than 2 FPGAs, it'd be complex to transfer signals between the FPGAs. For example, for a design that's partitioned on 4 FPGAs, it'll be complex to connect all the FPGAs together by wires. Instead, one of the solutions is to connect each FPGA by 2 other FPGAs as shown in figure (1).

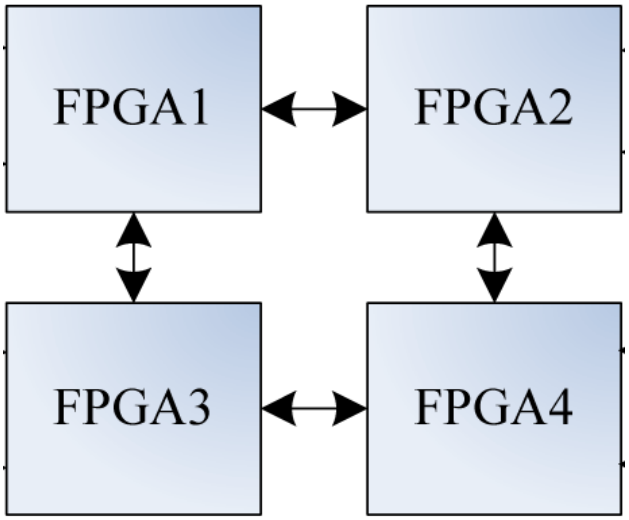

Figure 1: Multi-FPGA System

Considering the system in the figure, what if FPGA1 needs to communicate with FPGA4? It'll have to go through FPGA2 or FPGA3, which increases delays of the signals and subsequently decreases the operating frequency significantly.

1.3 Wireless Solution

The solution of previous issues is using wireless connection between FPGAs. If the FPGAs are connected wirelessly, the limited pins would not be an issue at all, as the signal connecting the 2 parts will be transmitted between the FPGAs wirelessly. If the RF chain and the antenna are provided, the used pins will not occupy the whole pins as in the wired connection, as the only needed pins are those which interface with the RF chain that starts with a DAC/ADC. Moreover, the Multiple-FPGAs Communication problem will not appear, as each FPGA is free to send its signals to any other neighboring FPGA. The challenges in this idea is to find the communication protocol that can transmit the data without affecting the design's main clock frequency or utilization of the FPGAs, so it has to provide high data rate and be implemented with a small FPGA utilization.

1.4 Problem Statement

To enhance the FPGA prototyping validation methods for very complex designs, a high data rate communication standard should be implemented with the least possible utilization of the FPGAs. Typically, this communication standard should support data rates in the range of Gigabits per second, and optimize its implementation to have the transmitter and receiver each has the minimum utilization of the used FPGA. The antenna and RF chain that work with this standard should be connected to the 2 FPGAs and provide minimum errors and minimum interference with the implemented hardware.

2. Literature Review

From the Problem statement it can be noted that the main objective is to find a low-utilization high-rate Communication standard, and find a suitable RF chain that can be used to convert the output digital samples into wireless waves.

2.1 Wireless Standards

Table 1 shows the most common high-rate Communication standards. The rate and FPGA utilization of each of them are discussed as follows.

Wireless Network Protocols	**Data rate**	**Carrier Frequency**
LTE	100 Mbps up to 300 Mbps	0.7 GHz to 2.6 GHz
Wi-Fi	11 Mbps up to 1.73 Gbps	2.4 GHz to 5 GHz
Bluetooth	1 Mbps up to 2 Mbps	2.4 GHz to 2.4835GHz
Wi-Gig	1000 Mbps up to 7 Gbps	60 GHz
Wireless HD	version 1.0 supports 4 Gbps version 1.1 supports 28 Gbps	60 GHz

Table 1: Common Communication Standards

The above table indicates that the most suitable standard to implement is the Wi-Fi as it has the required date rates in range of Gbps and suitable RF working in range from 2.4 GHz to 5 GHz unlike the other standards like LTE and Bluetooth which have lower data rates than the required and Wi-Gig which has very large carrier frequency, so it causes difficulties for the RF chain to fetch and work.

IEEE standard 802.11 (commercially known as Wi-Fi) has different Protocols and Data Rates. Table 2 compares the Wi-Fi protocols.

Protocol	Frequency	Channel Bandwidth	Maximum Data Rate
802.11ac wave2	5 GHz	80, 160 MHz	1.73 Gbps
802.11ac wave1	5 GHz	80 MHz	866.7 Mbps
802.11n	2.4 or 5 GHz	20, 40 MHz	450 Mbps
802.11g	2.4 GHz	20 MHz	54 Mbps
802.11a	5 GHz	20 MHz	54 Mbps
802.11b	2.4 GHz	20 MHz	11 Mbps

Table 2: Wi-Fi Protocols Summary

According to the specifications required, the decision of choosing which protocol of Wi-Fi will be chosen needs for much more information about all implementations made before as shown in the next sub-sections.

2.1.1 802.11ac:

Specifications:

- Up to 866 Mbps data rate.
- 80 MHz bandwidth.
- 5 GHz center frequency.

Implementations:

Paper 1: [1]

This paper presents a dual-band 2x2 Wi-Fi transceiver in 28nm bulk CMOS.

Area of RF + Analog die = 7.29mm^2

Paper 2: [2]

This paper aims to design and implement IEEE 802.11ac MAC controller in 65 nm CMOS.

The results are:

- This ASIC provides up to 45 Mbps throughput in a 20 MHz channel bandwidth mode and 267 Mbps throughput in an 80 MHz channel bandwidth.
- This transceiver is implemented in a 2.93 * 3.74 mm 2 chip area.

Paper 3: [3]

This paper proposes a dynamically reconfigurable end-to-end transceiver baseband that can switch between three popular OFDM standards, IEEE 802.11, IEEE 802.16 and IEEE 802.22. Three standards are implemented on a Xilinx Virtex 6 FPGA (XC6VLX240T).

.

2.1.2 802.11ad:

Specifications:

- Multi-Gbps communication.
- Data rates from 1 up to 7 Gbps.
- Center frequency 60 GHz.
- Frequency range from 58.32 to 70.2 GHz.
- 2.16 GHz maximum channel bandwidth.

Implementations:

Paper 1 [4]:

Specifications:

A transmitter and a receiver are implemented in this paper and partitioned into four FPGA boards (U1 through U4). The specifications and utilization are listed below.

The part number for U1 and U2 boards is LX330T while for U3 and U4 is XC95T.

- Base-band transmitter and receiver.
- Variable data rate up to 59 Mbps.
- Four Xilinx Virtex-5 boards for Tx, Rx, ADC and DAC.
- Bit rate could reach 2.5 Gbps for unlimited resources.
- 25-MHz and 45-MHz clock for Tx and Rx respectively.
- RF Tx/Rx front-end modules.
- ASIC design uses 220-MHz clock.
- High utilization and low data rate.

Utilization

- Inner Receiver (U2 and U4):
  - 52K out of 62K LUT - 84%.
  - 564 out of 640 DSK-48 units - 89%.
- ADC Board.
- DAC Board.
- Inner Transmitter: Single FPGA.
- Outer Transmitter: Single FPGA.
- RF front-end transmitter.
- RF front end receiver.

Paper 2: [5]

The transmitter and receiver of 802.11ad standard are implemented using both ASIC and FPGA approaches in this paper.

The ASIC design aspects are 3.12mm x 3.12 mm.
The achieved data rate is 2.3 Gbps.
The used clock frequency is 156 MHz's.
The FPGA utilization is summarized in table 3 and table 4.

FPGA board	Blocks
XC6VLX550T	Tx, SERDES, RX (Correlator, Channel estimation, CFO est. & comp., descrambler, HCS)
XC6VLX760	FFT, IFFT, FDE, Timing phase est. & comp., Feedback phase comp., Control PHY receiver
XC6VLX760	LDPC decoder

Table 3: Block allocation of the design on the boards

Board	Slices	DSP	Clock (MHz)
XC6VLX550T	41%	30%	68
XC6VLX760	49%	84%	82
XC6VLX760	35%	0%	84

Table 4: Utilization for each FPGA

Paper 3: [6]

A base-band and RF transceivers are implemented in this paper. The baseband required four Virtex-II FPGAs. The specifications and FPGA utilization are listed below.

Specifications:

- RF Transmitter and receiver modules.
- 60 GHz center frequency.
- 500 MHz bandwidth.
- Omnidirectional antennas.
- BPSK to 64-QAM modulation.
- 100 MHz clock frequency.
- 120 and 480 Mbps for BPSK and 16-QAM.

FPGA utilization

- Four Virtex-II boards.
- One board for the transmitter.
- 2 boards for the receiver.
- Small boards for ADC, DAC and baseband filtering.
- Transmitter board
    - 18K Flip Flops.
    - 12K Slices.
    - 41 Multipliers.
    - 11 RAM blocks.
    - 30% utilization.
- 125% utilization for the receiver.

2.1.3 802.11a:

Specifications:

- Center frequency can be 3.7 or 5 GHz.
- Occupied bandwidth is 20 MHz.
- Uses OFDM waveform.
- Data rate ranges from 6 to 54 Mbps.

Implementations:

Paper 1: [7]

A baseband and RF transceivers are implemented in this paper. The baseband part is implemented on Virtex-II FPGA boards. The design specifications are listed below. The FPGA utilization is shown in Table 5.

	Slices	Blocks
Transmitter	1115/33792 (3.3%)	10/144 RAMs (7%)
Receiver	1150/33792 (3.4%)	10/144 RAMs (7%)
Synchronizer	1150/33792 (3.4%)	18/144 Multipliers (12.5%)

Table 5: Utilization of FPGA

2.1.4 802.11n:

Specifications:

- 2.4 and 5 GHz center frequencies.
- 20 and 40 MHz bandwidth.
- Up to 288.8 and 600 Mbps.
- 4-MIMO streams.

Implementations:

Paper 1 [8]:

A baseband transceiver is fully implemented on Virtex-6 FPGA boards. Multiple Input Multiple Output (MIMO) technique is used in this paper. The design specifications and FPGA utilization is listed below.

- Full Baseband Transceiver.
- Center frequency: 2.4 GHz.
- Bandwidth: 20 MHz.
- Data rates is upper bounded to 216 Mbps.
- 40-MHz FPGA clock frequency.
- 20-MHZ Intermediate frequency.
- 14 and 12 bits DACs and ADCs run at 80 MS/s.
- XC2V6000-6 board for DSP.
- XC2V1000-4 board for each antenna pair.
- Both boards are fully utilized.

2.2 Transmission Method

2.2.1 Antenna and RF Chain

The task of choosing the RF chain is very important because the chosen chain must be suitable for the bandwidth and the data rate of the chosen standard to implement. USRP was suggested to be used as an RF chain in the project because it was available and tested before. USRP, which stands for Universal Software Radio Peripheral, is a range of software-defined radios designed and sold by Ettus Research and its parent company, National Instruments. USRP has many types, but the available board was B200, as shown in figure 2, which has the following specs:

1. RF coverage from 70 MHz – 6 GHz.
2. Flexible rate 12-bit ADC/DAC.
3. Up to 56 MHz of instantaneous bandwidth.

According to the previous specs the chosen standard must satisfy these specs to work properly.

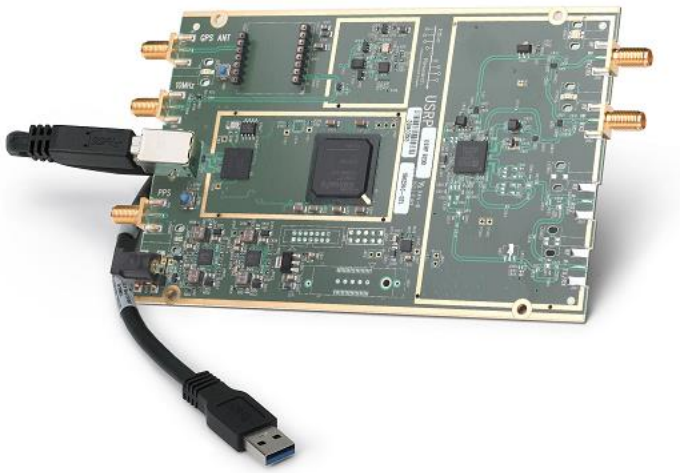

Figure 2 : USRP B200 board

2.2.2 Optical Transmission

Most of the Gigabit communication protocols work in very high carrier frequencies (60 GHz), so it's very hard to find an antenna and an RF chain that works at these frequencies, and since the communication will be short range (only between the 2 FPGAs so in the range of centimeters) the Optical methods of transmission should be considered.

2.2.2.1 Visible-Light Communication (VLC)

This method is simple. The Transmitter consists of some LEDs connected to the data lines, and if the bit is 1 the LED is on, if the bit is 0 the LED is off (On-Off Key). The Receiver on the other hand consists of Photodiodes that sense the LEDs' light and conduct current when the LEDs are on. This way the transmission can happen wirelessly without the need for antennas.

Disadvantages of VLC are that it only works in short ranges (not an obstacle in this application) and it can't provide Gigabit data rates, so another method should be considered.

2.2.2.2 Laser

Most FPGA kits have SubMiniature Version A (SMA) and Small Form-factor Pluggable (SFP) ports connected to some of the FPGA's pins. SFP port is a port that can be connected to an Optical fiber, and a laser-to-fiber coupler can be connected to the fiber to get Laser rays in and out of the fiber. Laser can provide Gigabit data rate easily, so if a standard is implemented on the FPGA and its output bits are connected to the SFP pins we can transmit the data using Laser. Similarly, the receiver can receive the data from the SFP port of the other FPGA kit. Upon searching for the laser-to-fiber coupler, couplers converting from SMA ports to laser directly were found, so these couplers can be connected to the SMA port directly without fibers.

2.2.2.3 Optical Fiber

Optical Fibers can support transmissions with data rates up to 10 Terabits per second, so they can be used instead of laser. In that case the connection will be wired but through an optical fiber and a communication standard not just simple wires.

2.3 Conclusion

The most suitable standard for our case is IEEE 802.11n; since the center frequency is low and could be 2.4 or 5 GHz, and it supports bandwidth 20 MHz and 40-MHz. The suitable mode if we want to use USRP B200 board is 40-MHz; since the RF daughterboard has a maximum bandwidth of 56-MHz, so even though 160-MHz bandwidth would give higher rate, the implemented standard would support 40-MHz due to the limited resources. [9]

Optical Transmission supports high rates, but it'd not aid in the Multi-FPGA connections issue discussed in chapter 1, hence, even though optical transmission methods are available, the wireless transmission is preferred for FPGA partitioning applications.

3. UART Wired FPGA Communications

Before implementing a wireless communication between 2 FPGAs, a simple wired communication is required to be implemented between FPGAs so that these techniques can be compared and get the advantages and disadvantages of both techniques.

UART protocol is chosen for wired communication. It stands for Universal Asynchronous Receiver / Transmitter. UART is a simple, half-duplex, asynchronous, and serial protocol. The transmitter converts parallel data from a controlling device into serial form, transmits it in serial to the receiver which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two FPGAs. FPGA transmits data asynchronously, which means there is no clock signal to synchronize the output of bits from the transmitting FPGA to the sampling of bits by the receiving FPGA.

Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the UART receiver knows when to start reading the bits. When the UART receiver detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate. Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). There is another bit that's called the parity bit, which describes the evenness or oddness of a number. The parity bit is a way for the UART receiver to tell if any data has changed during transmission. Figure 3 shows the frame of UART.

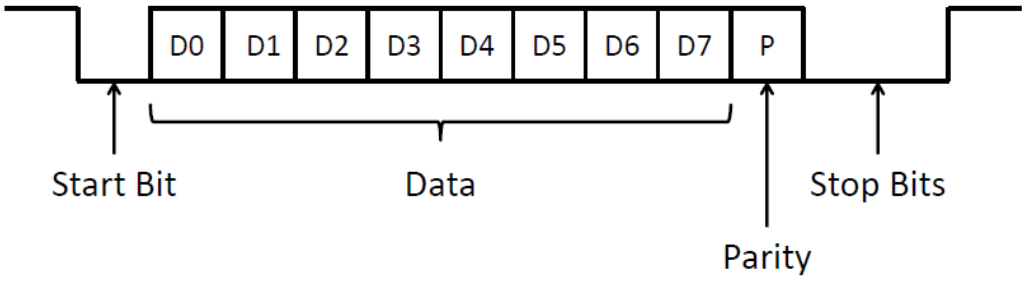

Figure 3: UART Frame

UART is implemented on Spartan-6 FPGA SP605 Evaluation Kit using Xilinx ISE tool with clock frequency 27 MHz.

3.1 Results

3.1.1 Transmitter

1. Utilization
    a. Slice Registers: 36 out of 54,576 (1%)
    b. Slice LUTs: 44 out of 27,288 (1%)
2. Power report is shown in figure 4.

On-Chip	Power (mW)
Clocks	1.11
Logic	0.06
Signals	0.07
IOs	1.93
Static Power	36.09
Total	39.26

Figure 4: power report of transmitter

3.1.2 Receiver

1. Utilization
    a. Slice Registers: 337 out of 54,576 (1%)
    b. Slice LUTs: 323 out of 27,288 (1%)
2. Power report is shown in figure 5.

On-Chip	Power (mW)
Clocks	6.60
Logic	0.30
Signals	1.55
IOs	0.49
BlockRAM/FIFO	14.76
8K BlockRAM	0.00
16K BlockRAM	14.76
Static Power	36.40
Total	60.10

Figure 5: power report of receiver

3. BER is calculated for 10 k sample and it is found less than 10^{-4}

3.2 Debugging

ChipScope is an embedded, software based logic analyzer. By inserting integrated logic analyzer (ILA) into the design and connecting them properly, signals can be monitored in the design. ChipScope provides convenient software based interface for controlling the integrated logic analyzer, including setting the triggering options and viewing the waveforms.

ChipScope is used to debug the UART implementation on FPGA as shown in figure 6.

Figure 6: ChipScope to analyze the output of Receiver

4. IEEE 802.11n PHY Specifications

The objective of the project is to implement a wireless, high-rate, low-utilization standard for communication between the FPGAs, and from chapter 2 it was clear that the chosen standard is Wi-Fi 802.11n [10]. The next step is to read the standard and understand the PHY layer specs, and then develop a high level MATLAB model of the transmitter and receiver to help in implementing them on the FPGAs. This chapter summarizes the specs stated by the standard, and the implementation will be discussed in next chapters.

4.1 Modes of operation

The standard specifies different modes of operation. First, note that Wi-Fi standard has many versions including 802.11a, 802.11n, 802.11ac, as discussed in chapter 2, so this version is different from 802.11a in some features, like the Channel Bandwidth and MIMO operation. For example, 802.11a version has 20 MHz channel BW, while 802.11n has 20 MHz and 40 MHz, so modes of operation are classified according to many subjects, some of them are discussed as follows:

4.1.1 Channel Bandwidth

The standard works at 20 MHz or 40 MHz channel BW as stated above, and since the objective is to maximize data rate, the 40 MHz BW mode is chosen for implementation.

4.1.2 Number of streams

As stated, this version supports the MIMO operation, which means that instead of having single transmitter and receiver chains, the transmitter and the receiver both have multiple parallel chains that send their signals to multiple Antennas and RF chains. Such a system would have a very high data rate. For example, if SISO system has 100 Mbps rate, then for a 4x4 MIMO system of the same standard the rate would reach 400 Mbps. Though MIMO operation increases data rates, for this effect to appear practically the MIMO channels must have minimum correlation, so it's used at relatively long distances. Since our channel is between FPGAs (a few centimeters long), then MIMO operation's effect would be negligible, not to mention its high complexity and utilization. Hence, the chosen mode of operation for implementation is SISO.

4.1.3 Compatibility

Since this is not the 1st version of Wi-Fi, it's expected to communicate with earlier versions in WLAN. Mainly, devices that operate with 802.11a version are called **Legacy** devices, while 802.11n version is called **High Throughput (HT)** version, hence, according to compatibility, modes of operation are as follows:

1. **Non HT**: Communicates with legacy devices only (802.11a operation).
2. **HT mixed**: Communicates with legacy and HT devices.
3. **HT green-field**: Communicates with HT devices only.

For compatibility with most Wi-Fi receivers, HT-mixed mode of operation is chosen for implementation.

4.2 PPDU Format

The objective is to only implement the Physical layer (PHY) of the standard, as it's enough to provide successful communication between FPGAs on bit level, but the standard describes PHY specifications for anyone to implement, so generally, PHY gets its inputs bits from a higher layer (MAC layer), and the data sent from the PHY layer as an output from the transmitter to the DAC is called PPDU. For implementation of Wi-Fi standard, one must follow the PPDU format specified in the standard.

PPDU formats differ according to the compatibility mode of operation. Figure 7 summarizes the formats for all 3 modes.

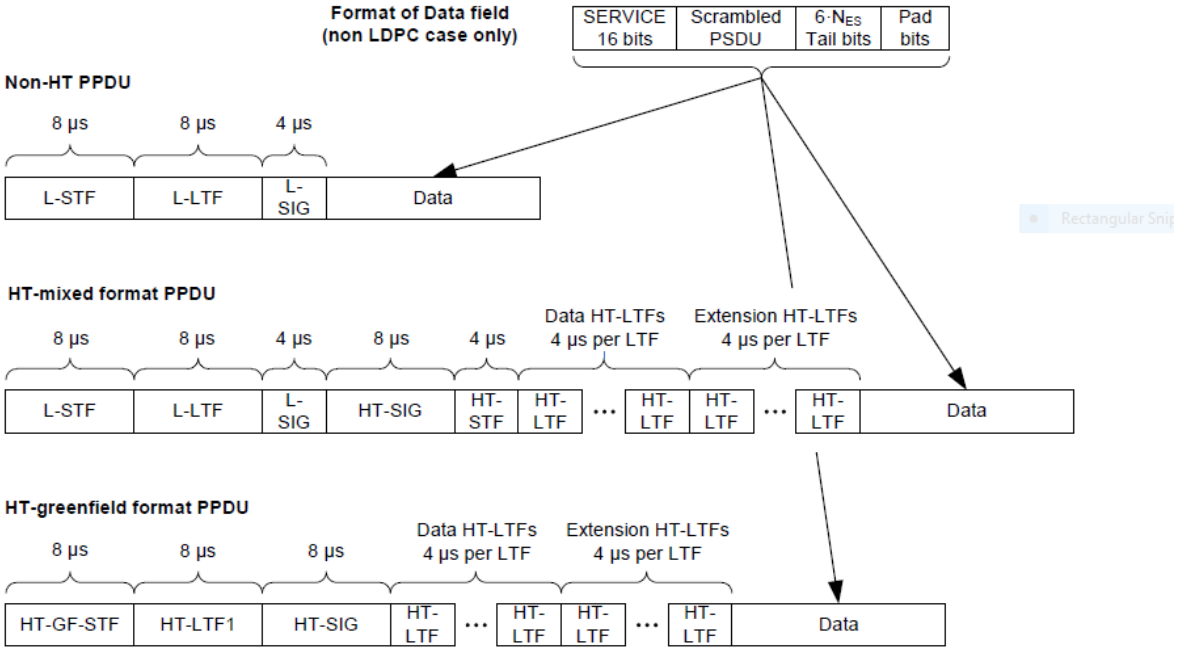

Figure 7: PPDU Formats

As the HT-mixed mode is to be implemented, the HT-mixed PPDU format is discussed. The elements of the PPDU are summarized in table 6.

Element	Description
L-STF	Legacy Short Training Field
L-LTF	Legacy Long Training Field
L-SIG	Legacy SIGNAL Field
HT-SIG	HT SIGNAL Field
HT-STF	HT Short Training Field
HT-LTF	HT Long Training Field
Data	Data Field

Table 6: Elements of the PPDU Format

Generally, STF and LTF fields are specific sequences transmitted to help the receiver in synchronization with the transmitter, to know the exact timing offset and to minimize the frequency offset between the transmitter and receiver carrier frequencies. More on this topic will be discussed in chapter 7 in the synchronization blocks implementation.

The SIGNAL Field carries information about the coding of the signal, and encoded in such a way that minimizes errors at the reception. The contents of the SIGNAL field are explained in detail in the standard.

The STF, LTF, and SIG fields are only sent initially, and then input data bits are encoded to form consequent OFDM symbols that transmit in the Data field. The encoding of the Data field is discussed in the next section.

4.3 Data Encoding

HT-mixed format and HT-greenfield format transmissions can be generated using a transmitter consisting of the following blocks:

1. *Scrambler:* scrambles the data to reduce the probability of long sequences of 0s or 1s, as they'd cause overflow errors at the end of the transmitter chain.

2. *Forward Error Correction (FEC) encoder:* encodes the data to enable error correction. An FEC encoder may include one of the following:
   a. A binary convolutional encoder followed by a puncturing device, and it'd be decoded by a Viterbi Decoder at the receiver.
   b. An LDPC encoder, and it'd be decoded by an LDPC Decoder

   Since the LDPC decoder is more complex (more utilization), the convolutional encoder, followed by a puncture, is chosen for implementation.

   The convolutional encoder, like the scrambler, is a circuit that encodes the input bits into 2 parallel bits that can be decoded back through a Viterbi decoder. It can be seen how its encoding process reduces the data rate in half, so puncturing is used to increase the rate by removing some bits in specific locations in the stream. Note that the removed bits can be recovered back at the receiver by the Viterbi decoder. The output rate of the puncturing device, relative to the input rate, can be 1/2, 2/3, 3/4, or 5/6, depending on the mode of operation.

3. *Interleaver:* Interleaves the bits of the spatial stream (changes order of bits) to prevent long sequences of adjacent noisy bits from entering the decoder. The Interleaver interleaves a specific number of bits depending on the mode of operation.

4. *Constellation Mapper:* Maps the sequence of bits in the spatial stream to constellation points (complex numbers). The modulation scheme used can be BPSK, QPSK, 16-QAM, or 64-QAM, depending on the mode of operation.

5. *Pilot Insertion:* For each 108 symbols coming out of the mapper, some zeros are inserted before, after, and in the middle of the symbols, and pilots are inserted at specific indices. Pilots are known values that are inserted to let the receiver estimate the channel by comparing their values to the received values at their locations. The output of this block is 128 symbols

6. *Inverse Discrete Fourier Transform (IDFT):* Converts a block of constellation points to a time domain block.

7. *Cyclic Prefix Insertion:* This is where the insertion of the cyclic shifts prevents Inter-Symbol Interference (ISI) and Inter-Carrier Interference (ICI). It simply prepends to the symbol a circular extension of itself.

The detailed operation and implementation of each of these blocks is discussed in the next chapter, but it can be seen that there are several modes of operation. The standard, though, only specifies 8 modes of operation for SISO, summarized in table 7.

MCS Index	Modulation	R	$N_{BPSCS(i_{SS})}$	N_{SD}	N_{SP}	N_{CBPS}	N_{DBPS}	Data rate (Mb/s)	
								800 ns GI	400 ns GI
0	BPSK	1/2	1	108	6	108	54	13.5	15.0
1	QPSK	1/2	2	108	6	216	108	27.0	30.0
2	QPSK	3/4	2	108	6	216	162	40.5	45.0
3	16-QAM	1/2	4	108	6	432	216	54.0	60.0
4	16-QAM	3/4	4	108	6	432	324	81.0	90.0
5	64-QAM	2/3	6	108	6	648	432	108.0	120.0
6	64-QAM	3/4	6	108	6	648	486	121.5	135.0
7	64-QAM	5/6	6	108	6	648	540	135.0	150.0

Table 7 : Modulation Coding Schemes

Modes of operation are organized by what's called a Modulation Coding Scheme (MCS) Index. As shown in table 7, for each MCS index the blocks work at different parameters, and as MCS index increases data rate increases and Bit-error rate (BER) increases, so the project will be implemented for all MCS indices to give high rates at good channel conditions and low errors at bad channel conditions.

It can be shown from the table that the highest achievable rate is 150 Mbps, and this is due to the limited BW of the available USRP boards (only 56 MHz) and the problems of having an RF chain that works at 60 GHz Carrier frequency. If such chains are available, 802.11ad standard would be the right choice for implementation and rates would reach multiple Gigabits per second, also if an RF chain with 160 MHz BW is available at 5 GHz carrier frequency, 802.11ac standard would be implemented, which could reach 600 Mbps for a SISO channel. However, changing the standard from 802.11n to 802.11ac is easy as both are versions of Wi-Fi, so it can be done easily when the chains are available.

A MATLAB High Model of the transmitter and receiver was developed for all 8 MCS indices, and reception was successful with zero errors for infinite SNR, which means the receiver's model reverses the transmitter's operations successfully. The model was used to verify that the RTL implementation.

The next chapters discuss the detailed operation and implementation of the transmitter and receiver, and the obtained results in terms of data rate and FPGA Utilization.

5. Transmitter

In this chapter, the transmitter implementation is presented, and the results are shown. First, a system model is shown to describe the block diagram and interactions between each block, and then each block's detailed operation and digital implementation are discussed.

5.1 System Model

The transmitter's implementation can be modeled by a series of blocks; each one performs one of the functions described by the standard, as discussed. In this section, a system level design is shown in figure 8, the requirements for proper integration of the blocks are presented, and the main digital implementation considerations are discussed, such as clock frequencies and communication between blocks.

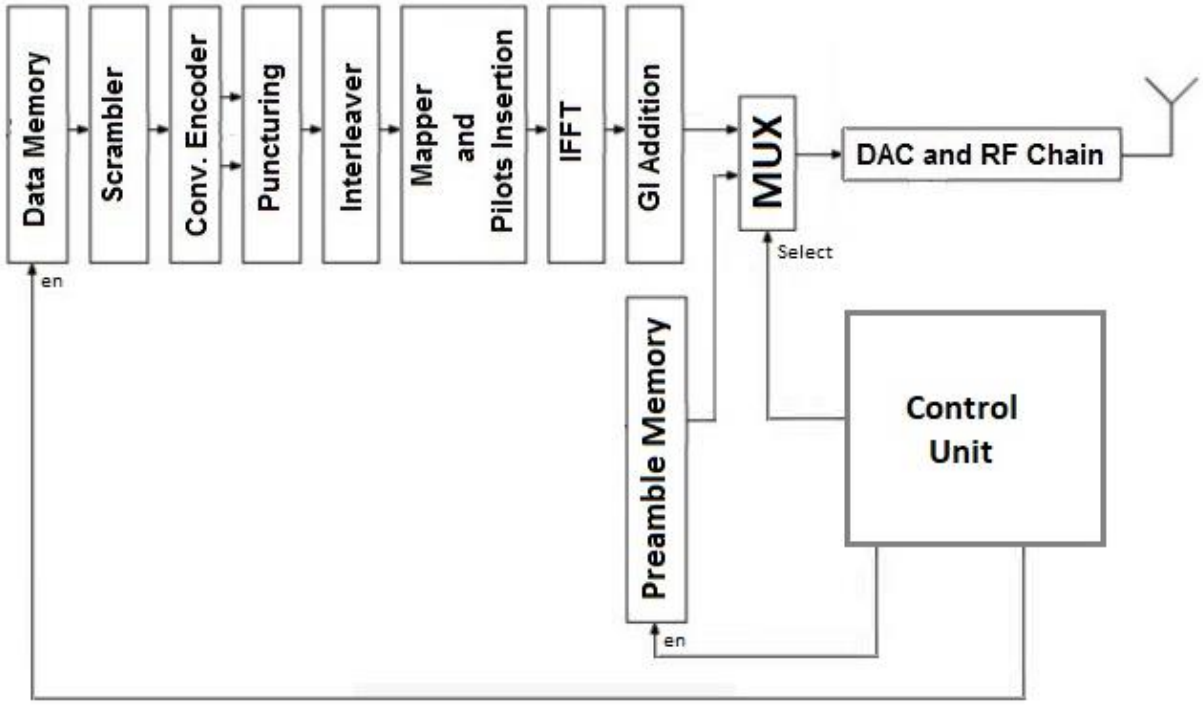

Figure 8: Transmitter Block Diagram

As shown in figure 8, generally the transmitter consists of the following blocks:

1. The data memory that has the bits that'll be transmitted to the receiver.
2. The digital chain that performs operations on the bits.
3. The Digital-to-Analog Converter (DAC) and the RF Chain needed to send the signal to the Antenna and propagate wirelessly.

This project covers the data memory and digital transmitter chain (baseband transmitter), while the DAC and RF chain will be implemented through the USRP board.

The main timing constraint specified by the standard is that, for 40 MHz bandwidth, the OFDM symbol's duration should be 4 µsec. in case of long GI, and 3.6 µsec. in case of short GI. Since the long GI OFDM symbol contains 128*1.25 = 160 samples, then each sample should take 25 ns, assuming a continuous stream of 24-bit samples at the output.

For this spec. to be satisfied, sample-domain blocks should work with a clock period of 25 ns, mainly, a 40 MHz clock, and for this stream to be continuous, some blocks will need to stop working each specific amount of time. This is discussed in detail as follows.

5.1.1 GI Addition Block:

This block's output should be a continuous stream of samples, and in case of long GI, for each 128 samples of the IFFT's output there should be 160 samples representing them, since both blocks work with the same clock frequency, then after each 128 samples the IFFT block should stop working for 32 clock cycles, and then continue for 128 cycles and so on. Note that for the IFFT block to stop without losing data, all the blocks before it should stop too, which means that the whole transmitter chain (except the GI block) should stop working each 128 sample-clock cycles for 32 sample-clock cycles. This can simply be implemented by a hold signal sent as a feedback from the GI addition block to all the blocks in the chain. When this signal is '1', the blocks stop working.

5.1.2 IFFT Block:

The IFFT block outputs 128 samples for 128 input samples, so it doesn't need to stop any blocks, yet its operations should be controlled be the hold signal coming from the GI block. The IFFT should work for 40 MHz clock, just like the GI block.

5.1.3 Mapper and Pilots Insertion Block:

This block's position is critical in terms of timing, as it transforms serial data bits into complex symbols which propagate with a different slower clock, so it's between 2 clock domains. The block's operation in brief is that it changes every set of bits into a complex symbol (based on the constellation used) and inserts pilots and zeros in their locations. For the block to operate properly inside the system, the following points must be considered:

1. The block is controlled by a bit-domain clock, so it must produce symbols every N_{BPSC} (Number of bits per subcarrier) cycles. For instance, in 16-QAM constellation, the output symbol should be generated each 4 bit-domain cycles.
2. At the zeros and pilots' locations, the input stream must be held, so the block controls the blocks before it and stops them from working when the locations occur.
3. The sample-domain clock is the bit-domain clock divided by N_{BPSC}, so the output of the block propagates with the sample-domain clock frequency, but since this is where clock-domain crossing occurs, a small 1-entry FIFO is inserted to pass the output synchronized with the sample-domain clock. The FIFO has only 1 entry as the input and output rates are equal, even though the clocks are different.

The block should work at the bit-domain clock frequency, so as a worst-case scenario, it should work at 240 MHz clock frequency, and output 64-QAM samples in sample-domain clock frequency.

5.1.4 Interleaver:

The interleaver is used to avoid having multiple bits in the same faded sub-carrier. The Interleaver's input and output bits are the same for each OFDM symbol, and they're also serial so for integration with the other blocks it should only handle the hold signal's control and work at the bit-domain clock frequency, specified by the used constellation.

5.1.5 Puncturing Block:

The Puncturing block's operation is simple, yet its implementation is critical as it'll need to hold the blocks before it to sustain a continuous stream of bits. To discuss this in more details, consider the case of R = 3/4, this means that for each 4 input bits there are 3 bits that'll continue to the output, and 1 bit will drop. Note that the input is 2 parallel bits so in 2 cycles there'll be 4 input bits, yet for 3 output serial bits they'll take 3 cycles to propagate. So, in brief, for R = 3/4, the puncturing block should work for 2 cycles and stop the input stream for 1 cycles, and in these 3 cycles the chosen 3 bits should propagate at the output pin. The block should work with the bit-domain clock frequency, it should hold the blocks before it at the right time for the right number of cycles, and it should perform well when controlled by the hold signal coming from the Pilot Insertion block and the GI Addition block.

5.1.6 Convolutional Encoder:

This block is simple and its implementation is in the standard. Its design considerations should only be to work at the bit-domain clock frequency and to perform well when controlled by the hold signal.

5.1.7 Scrambler:

The scrambler is similar to the encoder, its timing constraint is to work under the bit-domain clock frequency, and its operation shouldn't be corrupted by the hold signals.

5.1.8 Data Memory:

The memory is 1K*32-bit words, and its output is read by the bit-domain clock frequency divided by 32. Each bit-domain cycle, the read word is shifted and the shifted bit reaches the output port to enter the chain. Similarly, the memory is controlled by a hold signal and enabled by the chain enable signal, coming from the main control unit.

5.1.9 Control Unit:

The control unit's purpose is to output the preamble and signal fields before the data symbols reach the output. The implementation discussed in this project is based on the fact that for least utilization, only 1 Modulation Coding Scheme (MCS) will be implemented on the FPGA, and the codes are parameterized to change the working MCS when needed, but then the user has to implement again the new hardware. Hence, the preamble is fixed for each implementation, so it's stored in a preamble memory, and the control unit controls the flow through the following functions:

1. It reads from the preamble memory and directs its data to the output port.
2. It enables the transmitter chain at the right time, such that the first output sample appears after the last preamble sample by 1 sample-domain cycle.
3. It directs the chain's output to the output port when the GI block's first output sample is ready.

The control unit is a sample-domain block so it needs to work at 40 MHz frequency, and its operation won't be interrupted by any hold signal so its enable should be connected to the transmitter's enable signal.

Note that for correct initial conditions, each block should start working when the first sample of output of the block before it is ready to be read, and to make sure this happens, each block should be enabled by the one before it, and each block should enable the one after it when the output starts to appear.

By this point the design considerations, timing constraints, and the requirements for proper integration of the blocks are discussed. The following section discusses the digital implementation of each block in the chain.

5.2 System Components of Transmitter

5.2.1 Scrambler

All the data bits transmitted by 802.11n are scrambled using a frame synchronous 127 bits sequence generator. This Scrambler is used to randomize the service PSDU and pad the data patterns, which may contain long strings of binary 1s or 0s. The octets of PSDU are placed in the transmitted serial bit stream, bit 0 first and bit 7 last. The scrambler uses the generator polynomial S(x) as follows:

$$\mathbf{S(x) = x^7 + x^4 + 1} \quad (1)$$

The 127bit sequence generated repeatedly by the scrambler is (leftmost used first), 00001110 11110010 11001001 00000010 00100110 00101110 10110110 00001100 11010100 11100111 10110100 00101010 11111010 01010001 10111000 1111111, when the "all ones" initial state is used. While transmitting data, the initial state of the 802.11a scrambler will be set to pseudo random non-zero state. The contents of the SIGNAL field of the 802.11a are not scrambled. Figure 9 describes the scrambler.

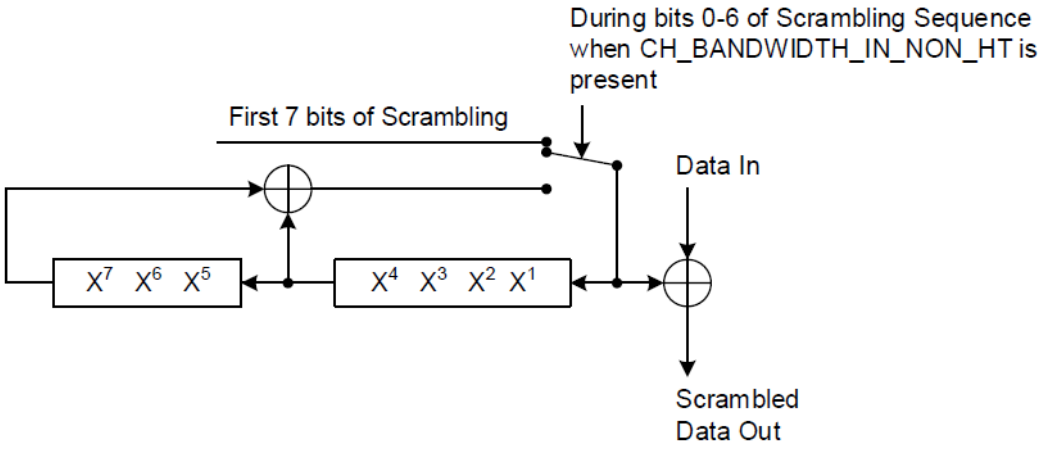

Figure 9 : Data scrambler

The implementation of scrambler consists of 7 shift registers and 2 XOR gates. It repeatedly generates a 127-bit sequence for a given pseudo-random initial state; each incoming data bit is XORed with the current bit in the 127-bit sequence as shown in figure 10.

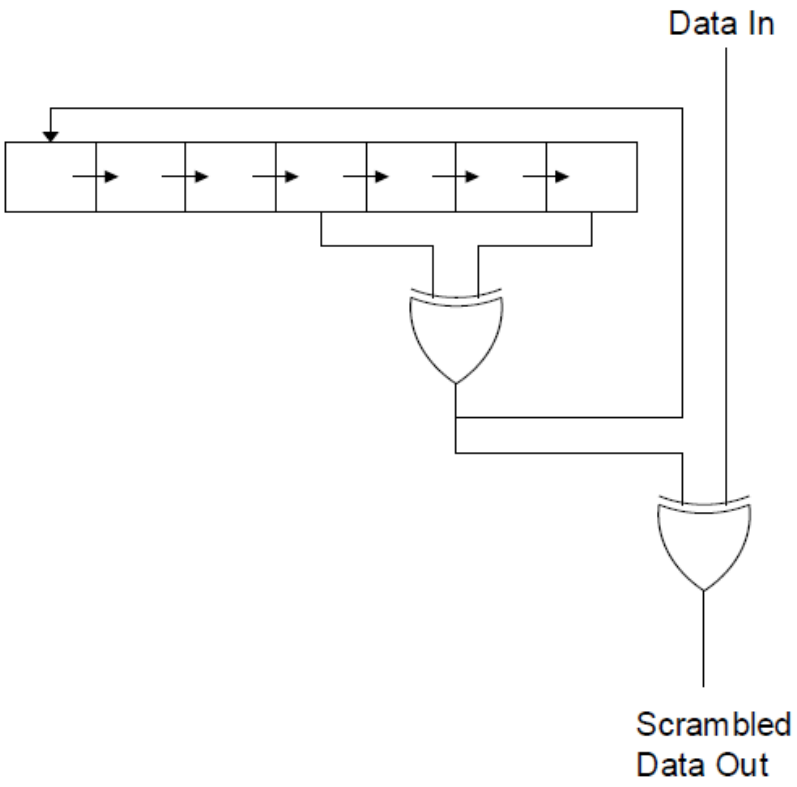

Figure 10: Implementation of scrambler

5.2.2 Convolutional Encoder and Puncturing

Encoding is used to add redundancy to the transmitted bits, redundancy adds extra information, and the extra information ensures that the data is received correctly.

The transmitted data shall be coded with a convolutional encoder of coding rate R = 1/2, 2/3, 3/4, or 5/6. In this standard the convolutional encoder shall use the industry-standard generator polynomials, $g_0 = 133_8$ and $g_1 = 171_8$, of rate R = 1/2, as shown in figure 11.

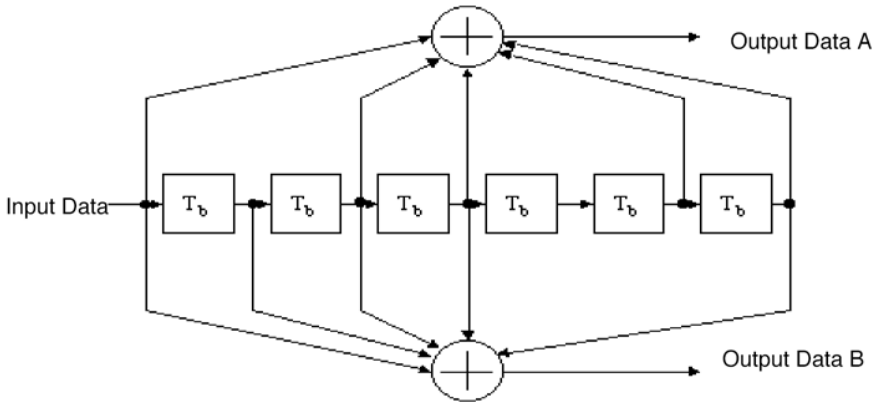

Figure 11: Convolutional encoder (k = 7)

The bit denoted as "A" shall be output from the encoder before the bit denoted as "B". Higher rates are derived from it by employing "puncturing". Puncturing is a procedure for omitting some of the encoded bits in the transmitter, thus reducing the number of transmitted bits and increasing the coding rate, and inserting a dummy "zero" metric into the convolutional decoder on the receiver's side in place of the omitted bits. The puncturing patterns are illustrated in figure 12. The decoding at the receiver's side is done using Viterbi decoder.

The implemented module has a shift register that takes a new bit and shift the exiting bits every positive clock edge. The output ports are connected as follows:

The implementation of the puncturing block is more complex; the implemented module includes two registers and synchronization flags between them. Concerning the complexity, it appears from the fact that inputs need to be taken in 3 cycles (3*2 parallel inputs = 6 bits) and the outputs need to be generated in 4 cycles, so a hold flag is implemented to hold all the previous blocks for a 1 clock cycle. Now The input comes in 3 cycles then it's held for 1 clock cycle (4 clock cycles) and the output is generated in these 4 clock cycles, hence, synchronization is achieved.

For example, consider puncturing with 3/4 coding rate. The input port delivers 6 bits in 3 cycles (as the output of the encoder has 2 bits width) and then they're stored into a register. When the register is full, these bits are loaded into another register with the required puncturing pattern, then the entire previous block are held (enable = 0) for one clock cycle. The output is generated by assigning the second register values to the output port. Figure 12 summarizes the operation for R = 3/4 and 2/3.

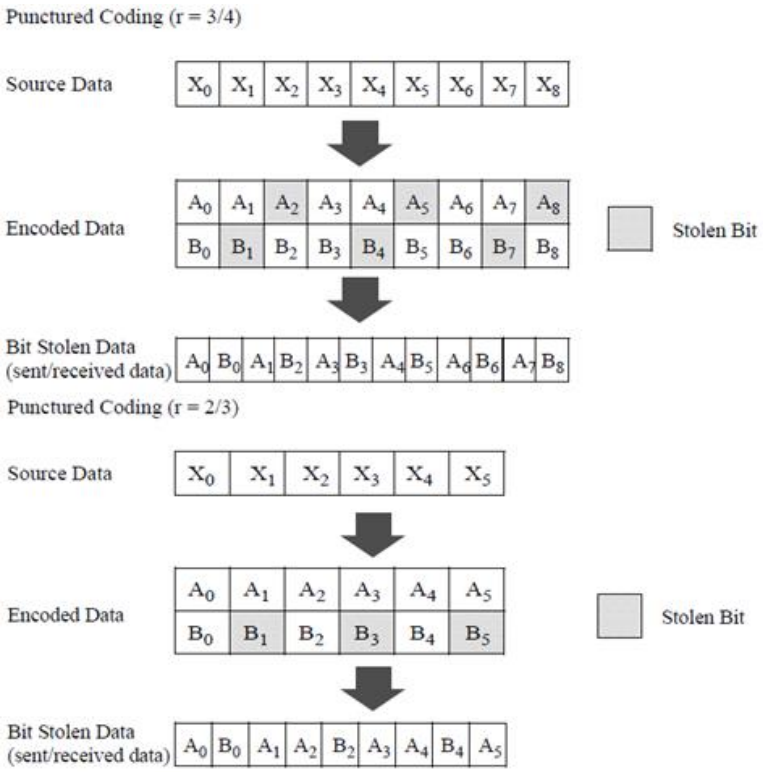

Figure 12: Puncturing patterns for R = ¾ and R 2/3

5.2.3 Interleaver

5.2.3.1 Operation

There are multiple types of channels, such as frequency selective channel or flat channel, and slow channel or fast channel. In frequency selective channels, the signal suffers from fading due to multiple paths. Interleaving increases resistance to frequency selective channel fading. When a part of channel bandwidth fades, interleaving ensures that the bit errors that would result from those subcarriers in the faded part of the bandwidth are spread out in the bit-stream rather than concentrated.

Interleaving is a process which disperses the positions of the data bits before transmission so that the corrupted information can be recovered at the receiver by rearranging the data.

Interleaver block size is corresponding to the number of bits in OFDM symbol, NCBPS. The Interleaver is defined by two permutations; the first permutation causes adjacent coded bits to be mapped onto non-adjacent subcarriers through equation (2). The second one causes adjacent coded bits to be mapped alternately onto less and more significant bits of the constellation and, thereby, long runs of low reliability (LSB) bits are avoided, it is explained in equation (3). The index of the coded bit is denoted by k.

$$\mathbf{i} = \left(\frac{N_{CBPS}}{16}\right) \mathbf{x}\ (\mathbf{k}\ \mathbf{mod}\ \mathbf{16}) + \left\lfloor \frac{k}{16} \right\rfloor, \mathbf{where}\ \mathbf{k} = \mathbf{0}, \mathbf{1}, \ldots\ldots, N_{CBPS} - \mathbf{1} \quad (2)$$

$$\mathbf{j} = \mathbf{s}\ \mathbf{x}\ \left\lfloor \frac{i}{s} \right\rfloor + \left(\mathbf{i} + \mathbf{NCBPS} - \left\lfloor \frac{16\ x\ i}{Ncbps} \right\rfloor\right) \mathbf{mod}\ \mathbf{s}, \mathbf{where}\ \mathbf{i} = \mathbf{0}, \mathbf{1}, \ldots\ldots, \mathbf{NCBPS} - \mathbf{1} \quad (3)$$

$$\mathbf{s} = \mathbf{max}\ (\mathbf{NCBPS/2}, \mathbf{1}) \quad (4)$$

5.2.3.2 Implementation

For ensuring that the Interleaver works with one bit per clock cycle, the architecture uses two memory modules simultaneously. One module is used to write into, while the other is used to read from. There is a third memory which stores the address generated by equations 2 and 3, so that the written data is written with this address and is read in order with a simple counter, as shown in figure 13.

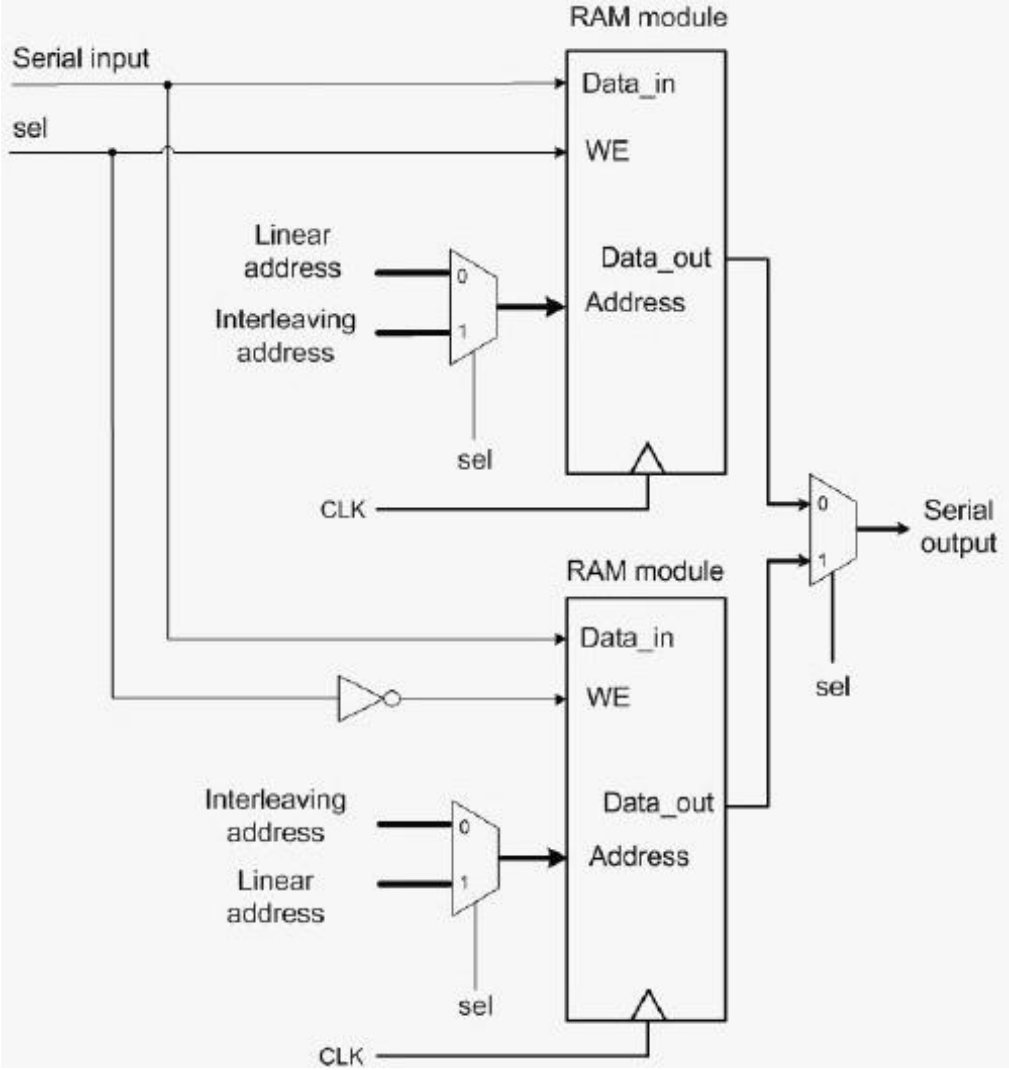

Figure 13: Implementation of Interleaver

5.2.4 Mapper and Pilots Insertion

5.2.4.1 Mapper

5.2.4.1.1 Operation

Each OFDM subcarrier shall be modulated, in this standard BPSK, QPSK, 16-QAM, or 64-QAM can be used, depending on the rate requested. The input encoded and interleaved bit stream shall be divided into groups of N_{BPSC} (1, 2, 4, or 6) bits then converted into complex numbers representing BPSK, QPSK, 16-QAM, or 64-QAM constellation points.

The conversion shall be performed according to Gray-coded constellation mappings, illustrated in figure 14 and figure 15; with the input bit, B_0, being the earliest in the stream.

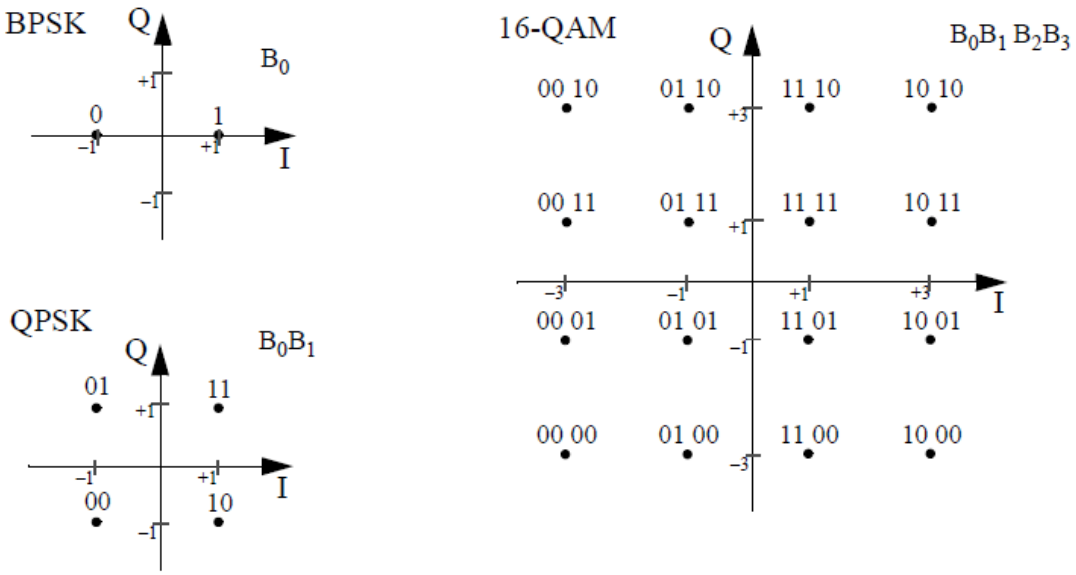

Figure 14: BPSK, QPSK, and 16-QAM constellation bit encoding

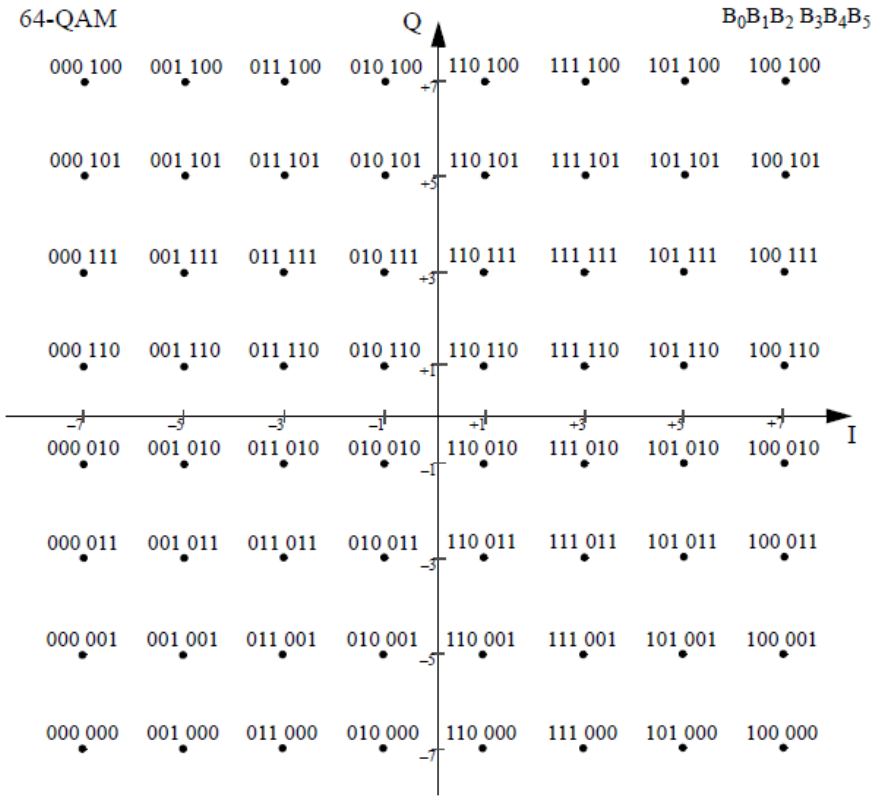

Figure 15: 64-QAM constellation bit encoding

In order to achieve an average power of 1 watt for all constellations, normalization shall be used, so the complex output (I + jQ) shall be normalized by multiplied by a normalization factor K_{MOD}, as described in equation 5.

$$\boldsymbol{d = (I + jQ) * K_{MOD}} \quad \textbf{(5)}$$

The normalization factor, K_{MOD}, depends on the base modulation mode, as prescribed in table 8.

Modulation	K_{MOD}
BPSK	1
QPSK	$1/\sqrt{2}$
16-QAM	$1/\sqrt{10}$
64-QAM	$1/\sqrt{42}$

Table 8: Modulation-dependent normalization factor K_{MOD}

5.2.4.1.2 Implementation

For example, consider 16-QAM constellation. The implemented module has a shift register which takes a new bit and shift the existing bits every positive clock edge, and a counter which increments each positive edge until the counter reaches the value 4. All the shift register bits are then converted into a symbol, and so on. Therefore, each 4 bit-domain clock cycles correspond to 1 sample-domain clock cycle. The same procedure is done in all constellations.

Concerning the normalization, it's handled during the conversion to symbols, meaning that for 64-QAM, if the desired symbol is $-7/\sqrt{42}$, then, instead of outputting -7 and multiplying it by $1/\sqrt{42}$, the value $-7/\sqrt{42}$ is generated at once to the output, which reduces utilization as no multiplier is used in any constellation.

5.2.4.2 Pilots Insertion

5.2.4.2.1 Operation

Pilots are known values that have specific locations in the OFDM symbol. Mainly, in 40 MHz channel BW, the data stream consists of time-domain OFDM symbols, each consists of 144 samples in case of short Guard Interval (GI), or 160 samples in case of long GI. Before inserting the GI, the OFDM symbol is 128 samples that are described in frequency domain as follows:

Index 0 to 5:	Zeros
Index 6 to 10:	Constellation symbols
Index 11:	Pilot 0
Index 12 to 38:	Constellation symbols
Index 39:	Pilot 1
Index 40 to 52:	Constellation symbols
Index 53:	Pilot 2
Index 54 to 62:	Constellation symbols
Index 63 to 65:	Zeros
Index 66 to 74:	Constellation symbols
Index 75:	Pilot 3
Index 76 to 88:	Constellation symbols
Index 89:	Pilot 4
Index 90 to 116:	Constellation symbols
Index 117:	Pilot 5
Index 118 to 122:	Constellation symbols
Index 123 to 127:	Zeros

Pilots' values are [1, 1, 1, -1, -1, 1] from pilot 0 to pilot 5, and they rotate left each OFDM symbol. Pilots are inserted to monitor the channel response variations with time through measuring their values in the receiver, while zeros are inserted to minimize adjacent channel interference as stated before.

5.2.4.2.2 Implementation

The bit-domain output of the mapper is inserted on 1 input port of a 4-to-1 MUX, zeros are inserted on another input port, and output of the pilots register is on the third port while the fourth is unused, as shown in figure 16.

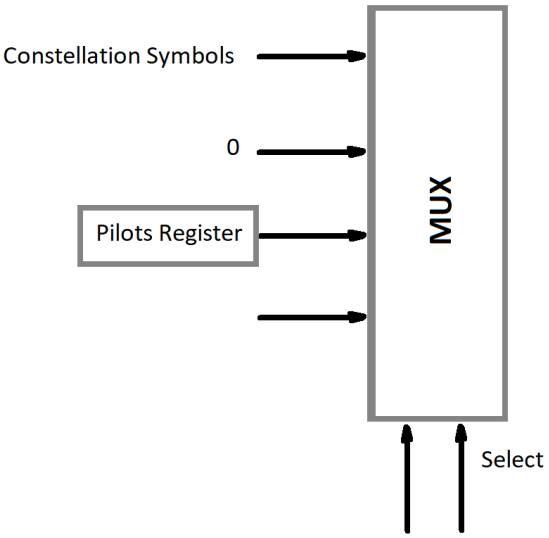

Figure 16: Pilots Multiplexer

The Pilots Register is a simple 6-entries shift register that holds the initial values of the pilots, and rotates its contents each OFDM symbol. Selection of the MUX is controlled by a simple counter that increments each N_{BPSC} bit-domain clock cycles. When the counter's value reach zeros' or pilots' index, the zeros/pilots are chosen for the MUX/s output, and all the previous blocks are held with a 1-bit hold signal until the counter returns to a symbol index.

The MUX's output is connected to a 1-entry FIFO. The FIFO has a read and write clock. The write clock is the bit-domain clock, and the read clock is the sample-domain clock. The inputs and outputs of the FIFO propagate with the sample-domain clock frequency, but the FIFO's output is synchronized with the sample-domain clock. The FIFO's existence is necessary to prevent timing violations and functional errors that occur in clock-domain crossing.

5.2.5 IFFT

The next step in encoding the signals is transforming the symbols from frequency domain to time domain through 128-point Inverse Discrete Fourier Transform operation. To implement the operation in hardware, the **Fast Fourier Transform** (FFT) algorithm is used. In the next pages, the FFT algorithm is discussed, and then a hardware implementation of the algorithm is presented.

5.2.5.1 FFT Algorithm

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{\frac{j2\pi nk}{N}} \qquad (6)$$

Equation 6 represents the Inverse Discrete Fourier Transform (IDFT). It can be seen that in order to compute 1 sample in the time domain, N complex multiplications and N-1 complex additions must be done, which means that for the transform to be computed, N^2 complex multiplications and N(N-1) complex additions must be done, which is computationally inefficient.

Complex Multiplications	Complex Additions	Real Multiplications (Number of Multipliers)	Real Additions (Number of Adders)
N^2	N(N-1)	$4N^2$	$N(N-1) + 2N^2$
16384	16256	65536	16512

Table 9: Discrete Fourier Transform Complexity

Table 9 shows an approximate utilization of the IDFT hardware at N = 128. James Cooley and John Tukey could reduce the complexity of IDFT implementation from $O(N^2)$ to $O(N\log_2 N)$ using the Fast Fourier Transform Algorithm, which reduces utilization dramatically [11].

Note that while equation 6 represents Inverse Discrete Fourier Transform, equation 7 represents Discrete Fourier Transform. The difference is in the exponent signs and the division by N, so FFT algorithm is discussed, and then it's modified to perform IDFT instead of DFT.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}} \qquad (7)$$

The FFT Algorithm is mainly based on decomposing N-point DFT into 2 N/2-point DFTs, so if N is an integer power of 2 (like the case of interest where N = 128) then N-point DFT can be decomposed into 2 N/2-point DFTs, which in turn can be decomposed into 4 N/4-point DFTs. and so on until 2-point DFT is reached.

The method of decomposing N-point DFT into 2 N/2-point DFTs is performed either by Decimation in time, or in frequency. Decimation-in-frequency method has known small-utilization hardware architecture, so it'll be discussed here, while Decimation-in-time method will not be covered in this Thesis.

In Decimation-in-frequency FFT algorithm, the decomposition is based on decomposing the output sequence into successive smaller sequences. Considering an N-point DFT where N is an integer power of 2, the even frequency samples can be obtained as follows:

$$\because X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}}$$

$$\therefore X(2k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi n(2k)}{N}} = \sum_{n=0}^{\frac{N}{2}-1} x(n) e^{\frac{-j2\pi n(2k)}{N}} + \sum_{n=\frac{N}{2}}^{N-1} x(n) e^{\frac{-j2\pi n(2k)}{N}}$$

Setting n = n – N/2 in the second summation

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) e^{\frac{-j2\pi n(2k)}{N}} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right) e^{\frac{-j2\pi\left(n+\frac{N}{2}\right)(2k)}{N}}$$

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) e^{\frac{-j2\pi n(2k)}{N}} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right) e^{\frac{-j2\pi n(2k)}{N}} e^{-j2\pi k}$$

$$\boldsymbol{X(2k) = \sum_{n=0}^{\frac{N}{2}-1} (x(n) + x(n+N/2)) e^{\frac{-j2\pi n(k)}{N/2}}} \qquad \textbf{(8)}$$

From equation 8, even samples can be obtained from N/2-point DFT of the addition of the first and second half of the input signals.

Similarly, the odd frequency samples can be obtained as follows:

$$\because X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}}$$

$$\therefore X(2k+1) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi n(2k+1)}{N}}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(n) e^{\frac{-j2\pi n(2k+1)}{N}} + \sum_{n=\frac{N}{2}}^{N-1} x(n) e^{\frac{-j2\pi n(2k+1)}{N}}$$

Setting n = n – N/2 in the second summation

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} x(n) e^{\frac{-j2\pi n(2k+1)}{N}} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right) e^{\frac{-j2\pi\left(n+\frac{N}{2}\right)(2k+1)}{N}}$$

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} x(n) e^{\frac{-j2\pi n(2k+1)}{N}} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right) e^{\frac{-j2\pi n(2k+1)}{N}} e^{-j\pi(2k+1)}$$

$$\boldsymbol{X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1}((x(n) - x\left(n+\frac{N}{2}\right))e^{\frac{-j2\pi n}{N}})e^{\frac{-j2\pi n(k)}{N/2}}} \quad \textbf{(9)}$$

From equation $\boldsymbol{X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1}((x(n) - x\left(n+\frac{N}{2}\right))e^{\frac{-j2\pi n}{N}})e^{\frac{-j2\pi n(k)}{N/2}}}$ (9), odd samples can be obtained from N/2-point DFT of the subtraction of the first and second half of the input signals, and multiplying them by $e^{\frac{-j2\pi n}{N}}$, which's called the **Twiddle factor**.

Equation 8 and 9 show how N-point DFT is decomposed into 2 N/2-point DFTs, and when N is an integer power of 2, the decomposition continues until it reaches 2-point DFT, which is only addition and subtraction. This can be shown in detail as follows.

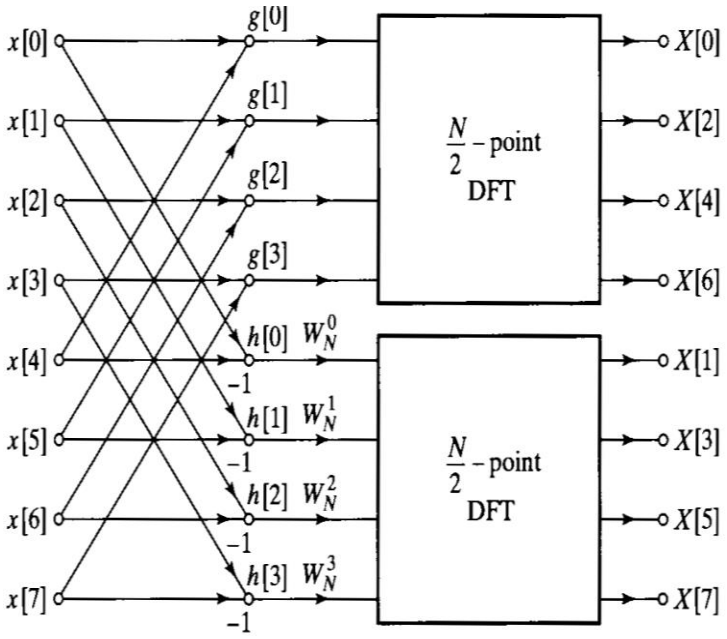

Figure 17: Signal flow graph of the first stage of Decimation-in-frequency FFT

Figure 17 shows how even and odd signals can be obtained from N/2-point DFT for N=8. Mainly, the input signals are added and subtracted, and the subtracted results are multiplied by the twiddle factor, where as $W_N^k = e^{\frac{-j2\pi k}{N}}$.

Similarly, each N/2-point can be decomposed into 2 N/4-point DFTs with additions, subtractions, and multiplications by twiddle factors, which can be shown in figure 18.

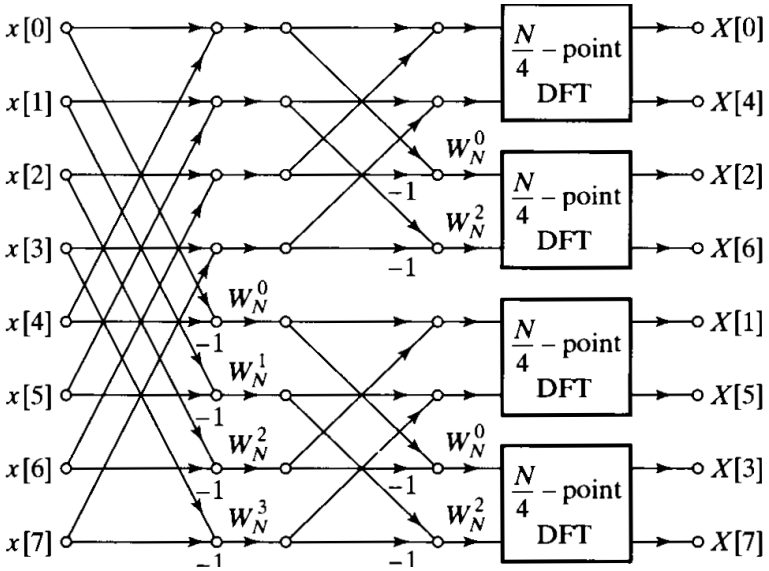

Figure 18: Signal flow graph of the first 2 stages of Decimation-in-frequency FFT

The procedure can be repeated for any power-of-2 number of points. Figure 19 shows the signal flow graph of 8-point FFT algorithm, and for the case of N = 128 the graph will be extended to have 4 more stages, as 128-point DFT will be decomposed into 64-point DFTs, 32-point DFTs, and all the way down to 2-point DFTs which are only additions and subtractions.

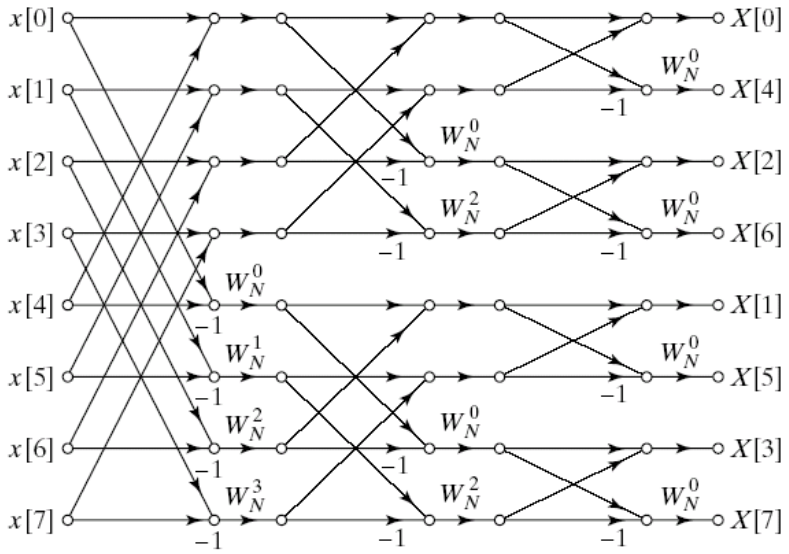

Figure 19: 8-point FFT signal flow graph

Note the output sequence's order in 8-point FFT. Generally, the output of FFT algorithm is in **bit-reversed** order, which is an ascending order in binary but written from left to write, so instead of 000, 001, 010, ... the order is 000, 100, 010, ... which means that the output needs to be reordered before inserting the **Cyclic Prefix** in the transmitter, so the reordering procedure is done in the **GI Addition block**.

Considering the computational complexity of FFT algorithm, for each stage there are N complex additions/subtractions and N/2 complex multiplications, and for power-of-2 N-point FFT there are $\log_2 N$ stages, so the complexity can be summarized in table 10.

Complex Multiplications	Complex Additions/ Subtractions	Real Multiplications (Number of Multipliers)	Real Additions / Subtractions (Number of Adders/ Subtractors)
$(N/2)\log_2 N$	$N \log_2 N$	$2N \log_2 N$	$N \log_2 N + N \log_2 N$
448	896	1792	1792

Table 10: Fast Fourier Transform Complexity

The numerical values in the table represent utilization at N = 128, and by comparison with table 9 it can be shown how FFT algorithm dramatically decreased the number of operations required for N-point DFT, and therefore it's easily implemented in hardware and software and its computational time is relatively small, hence the name Fast Fourier Transform.

As discussed earlier, this algorithm is derived for DFT operation, so it needs modification to work for IDFT case. Simply, the twiddle factors will be $W_N^k = e^{\frac{j2\pi k}{N}}$ instead of $W_N^k = e^{\frac{-j2\pi k}{N}}$, and since N is a power of 2 the division by N will only be arithmetic shift right by $\log_2 N$ times.

5.2.5.2 FFT Hardware Architectures

There are many hardware architectures that perform the FFT algorithm, but most of them share the idea that N-point FFT consists of $\log_2 N$ stages, each stage performs the additions, subtractions, and multiplications by the twiddle factors shown in figure 20.

Since the main goal of the project is to have least possible utilization, **Radix-2 Single-path Delay Feedback** (R2SDF) architecture was chosen for this block. The explained FFT algorithm is called **Radix-2 FFT** as each step divides the DFT number of points by 2 (which is the simplest algorithm), and this architecture is based on having the signals propagate in a single path that has a delayed feedback. [12]

Considering the case of N = 8, the FFT processor will have 3 stages. In R2SDF, the first stage consists of 3 blocks controlled by a control unit as shown in figure 20.

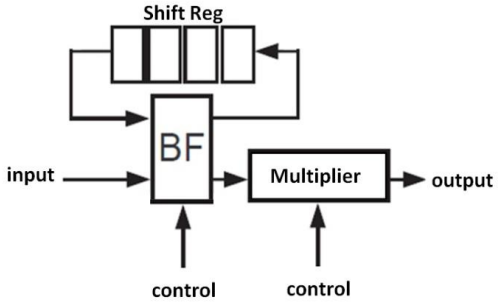

Figure 20: R2SDF Stage Block Diagram

Assuming that inputs enter serially in order from x[0] to x[7], mainly there are 2 modes of operation:

Mode 1:

1. The inputs are passed from the **Butterfly** block (BF) to the **Shift Register**.
2. The Shift Register's output passes through the Butterfly to the **Multiplier**.
3. The Multiplier multiplies its inputs by the twiddle factors and pass them to the output port of the stage.

As inputs enter the stage, mode 1 starts and lasts for 4 clock cycles, so by its end the Shift Register will be filled with signals x[0] : x[3], and the output is a trash (only initially).

Mode 2:

1. The Butterfly block performs the butterfly operation shown in figure (3), which is the addition and subtraction of its inputs. At this mode, the BF's 2 input ports are the input signals x[0]:x[3] (from the Shift Register's port) and the input signals x[4]:x[7] from the input port, so the BF adds its 2 inputs and passes them to the multiplier's input port, and the BF also subtracts its 2 inputs and passes them to the shift register's input port.
2. The shift register shifts the subtracted results by 4 clock cycles and outputs the inputs x[0]:x[3] to the BF.
3. The multiplier multiplies the inputs by 1 (passes them unchanged).

Mode 2 lasts for 4 clock cycles, and from figure 17 it can be seen that the output has the values of X[0], X[4], X[2], X[6], all propagate serially on the output port.

After 4 clock cycles the control unit goes back to mode 1, and then the new inputs will fill the shift register while the subtracted inputs will get out of the shift register and get multiplied by the twiddle factors. The output then will have the values of X[1], X[5], X[3], X[7] as shown in figure 17.

That was for stage 1 in the 8-point FFT. For stage 2, the operation will be the same except that each mode will last for 2 cycles instead of 4, and the shift register's size will be 2 instead of 4. Similarly, it can be seen that a 128-point FFT consists of 7 stages connected serially with sizes 64, 32, 16, 8, 4, 2, 1, whereas size represents the shift register's size, the number of cycles for which each mode lasts, and the number of twiddle factors. Note that for stages to operate properly, each stage should start working when the first sample of the previous stage's output becomes ready, which can be controlled by an enable signal for each stage.

The design of each block is as follows:

1. Shift Register:

   The basic implementation of any shift register is simply some registers connected in series, but since the main goal is reducing utilization, a memory-based implementation was done. As shown in figure 12, for a size of 4, a memory of size 4 words is implemented, with write address initialized by 0 and read address initialized by 1, and both addresses increment each cycle. After 3 cycles, the read address reaches the location of the first sample, and the sample gets out of the memory to the output port after 1 cycle, which performs the needed shift. The memories are implemented by memory cells in the FPGA (e.g. RAMB36E1) which are rarely used in the designs, so using these cells in the project will reduce its effect on the tested design's utilization.

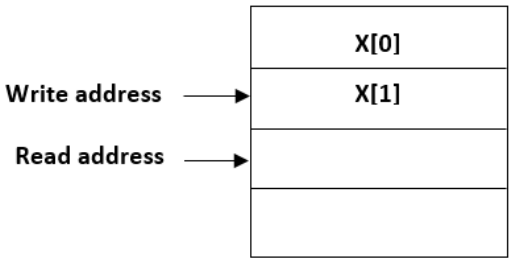

Figure 21: Shift Register Implementation

2. Butterfly:

   The butterfly's 2 outputs are connected to two 2-to-1 MUXs that choose based on the mode; they either choose the inputs or the results of addition and subtraction.

3. Multiplier:

The multiplication is generally one of the operations that have very large utilization, so the straightforward binary multiplier will not be considered as utilization needs to be as small as possible. Since all multiplications are between variable inputs and constant twiddle factors (which are complex exponentials), then the **Rotational CORDIC** will be the most suitable implementation for this block. CORDIC (Coordinate Rotation Digital Computer) is a block that rotates complex inputs by a known phase through a series of pipelined stages. CORDIC is based on the following:

If $Y = Xe^{j\theta}$ whereas X & Y are complex, then

$$\begin{pmatrix} Y_R \\ Y_I \end{pmatrix} = \begin{pmatrix} X_R \\ X_I \end{pmatrix} \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix}$$

$$\begin{pmatrix} Y_R \\ Y_I \end{pmatrix} = cos(\theta) \begin{pmatrix} X_R \\ X_I \end{pmatrix} \begin{pmatrix} 1 & -tan(\theta) \\ tan(\theta) & 1 \end{pmatrix}$$

So, if $\theta=\tan^{-1}(0.5)$ then

$$\boldsymbol{Y_R = cos(\theta) * \left(X_R - \frac{X_I}{2}\right)}$$

$$\boldsymbol{Y_I = cos(\theta) * (\frac{X_R}{2} + X_I)} \quad \textbf{(10)}$$

The division by 2 is only shifting right, and the additions and subtractions occupy small utilization. For the multiplication by cos(θ), its large utilization can be avoided using the fact that it's a constant multiplication, so it can be performed by **Multiple Constant Multiplication** (MCM) method. MCM method simply says that, for instance, in order to multiply x*0.75, first convert the constant to its binary fixed point form, which is 0.11, then 0.11 can be expressed as 0.1+0.01 so: $x * 0.75 = x * (0.11)_2 = x * (0.1)_2 + x * (0.01)_2 = x/2 + x/4$
And similarly, any constant multiplication can be converted into a summation of shifted values of the input.

To this point, it was shown how a complex variable can be rotated by a known phase of $\tan^{-1}(0.5)$ using only adders, subtractors, and shifters. The method can expand to rotate by $\pm \tan^{-1}(0.5)$, by adding a control signal that converts adders to subtractors and vice versa, so now the block rotates by 2 known angles using 1-bit control signal. To expand the block to rotate by many angles, it'll simply be a pipeline of multiple stages each one rotates by $\pm \tan^{-1}(2^{-k})$, so for 2-stages CORDIC, the first stage rotates by $\pm \tan^{-1}(0.5)$ and then the second one rotates by $\pm \tan^{-1}(0.25)$, so the rotated angles can be $\pm 40.6^{\circ}$ or $\pm 12.53^{\circ}$ depending on the control signals. The constant multiplication will be $\prod cos(\theta_k)$ $whereas$ $\theta_k = tan^{-1}(2^{-k})$ for k = 1, 2, 3, ..., m for m-stages CORDIC, and the constant multiplication will only be after the final stage. Figure 22 shows a single stage of the CORDIC multiplier.

As the number of stages increases, accuracy increases, but utilization and latency increase as well, so it was verified by MATLAB simulations that 11 stages are sufficient for rotating the inputs by the twiddle factors in all stages of the FFT. The control signals needed for all twiddle factors were generated by MATLAB and stored in a ROM, and the control unit reads them from the ROM and passes them to the multiplier.

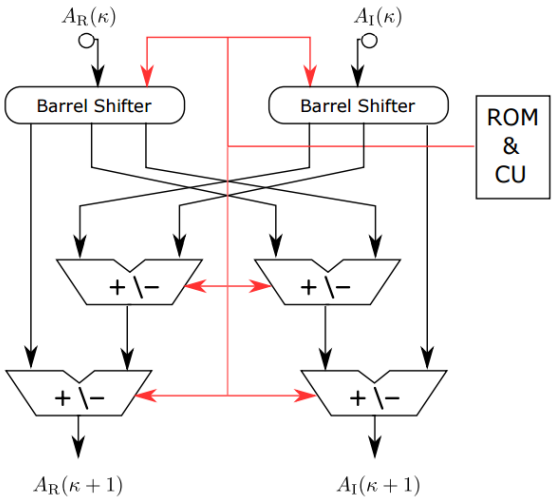

Figure 22: CORDIC Stage

4. Control Unit:

The control unit is simply a counter that switches between mode 1 and mode 2 and generates the addresses of the ROM control signals at mode 1 to pass them to the multiplier.

All blocks are parameterized to work in any stage, and the stage is parameterized to work as the third, fourth, fifth, sixth, or seventh stage, while the first and second stage are so simple that they're implemented separately without CORDIC. The FFT can be further expanded to operate as 256-point or any power-of-2 number of points; the only modification needed would be to increase the CORDIC stages to give good accuracy for the new stages' twiddle factors.

Considering the modification of FFT to operate as an IFFT, 2 changes are needed:

1. The CORDIC unit will change its operation to add/subtract instead of subtract/add as the angles' sign will be inverted.
2. The division by N will be implemented by shifting the results of each stage by 1 bit, and for causing lease errors; the result of shifting is rounded depending on the shifted bit.

The code is written in Verilog and parameterized to work as an FFT or an IFFT with any number of integer and fraction bits. The Signal-to-Quantization-Noise Ratio (SQNR) for the IFFT reached 60 dB for 24-bit inputs: 12-bit real and 12-bit imaginary, each contains 2 bits to represent the integer part and 10 bits for the fraction part.

The remaining steps in the transmitter are to bit-reverse the IFFT output and to add the cyclic prefix, and they're both included in the GI Addition block as mentioned earlier, and then the signal will be converted to analog and up-converted to be sent wirelessly to the receiver.

5.2.6 Guard Insertion

5.2.6.1 Operation

Orthogonal frequency division multiplexing (OFDM) is a transmission technique for wideband digital communications; the OFDM spectrum consists of many closely spaced orthogonal carriers. Orthogonality of carriers prevents interference between the closely spaced overlapping carriers. In high data rate communication, the time duration is decreases; this gives rise to self-interference due to multipath delay spread which is decoded incorrectly at the receiver. To avoid ISI, keep time duration greater than the maximum delay of the channel [13].

Guard interval is provided before the data period given by IFFT so that the ISI occurs in the guard interval which can be removed afterwards and the data can be retrieved, this makes high level of robustness against multipath delay spread of OFDM system. The guard period gives time for multipath signals from the previous symbol to decay before the information from the current symbol is gathered.

Cyclic Prefix (CP) is the insertion of the last portion of the OFDM symbol inside the Guard Interval. The CP is inserted to extend the periods of the subcarrier sinusoids in time domain to prevent Inter-Carrier Interference (ICI), as shown in figure 23.

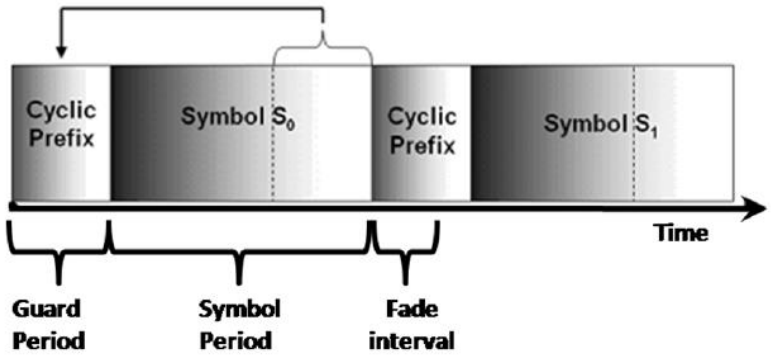

Figure 23: Cyclic prefix insertion

The relative length of the cyclic prefix depends on the ratio of the channel delay spread to the OFDM symbol duration. Therefore, there are two benefits of using a cyclic prefix:

- The CP isolates different OFDM blocks from each other when the wireless channel contains multiple paths.
- The CP turns the linear convolution with the channel into a circular convolution.

5.2.6.2 Implementation:

For ensuring the guard insertion works with one symbol per clock cycle, the architecture uses two memory modules simultaneously. One module is used to write over, while the other is used to read from, and vice versa. The IFFT bit-reverses the data in the OFDM symbol, that's why guard insertion stores the data with bit-reversed address and reads in order with a simple counter. When the data is read, the read address starts from either 112 in case of short GI, or 96 in case of long GI. When the CP is outputted, a hold signal is outputted from the GI to the entire previous chain as feedback to stop the input stream until all the CP's samples are outputted from the transmitter.

6. Receiver

6.1. System Model

The receiver's implementation is modeled as shown in figure 24, in this section the importance of each block, its digital implementation, and how they're connected together, including clock domains and synchronization between blocks, are discussed.

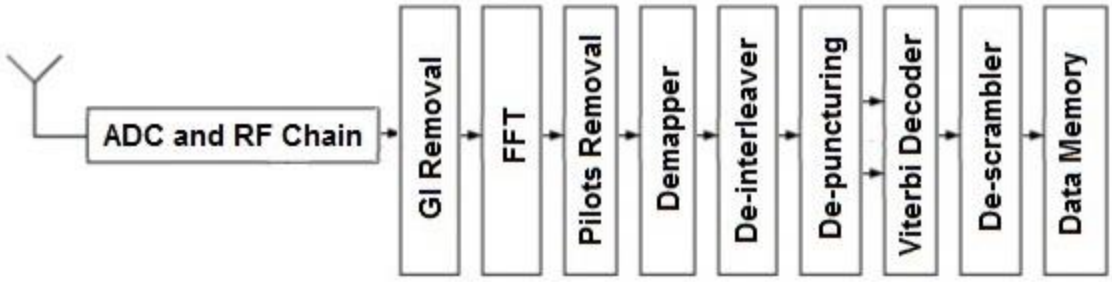

Figure 24: Receiver Block Diagram

As shown in figure 24; generally the receiver consists of the following:

1. The data memory that will receive the transmitted bits.
2. The digital chain that performs operations on the bits.
3. Synchronization blocks (will be discussed later).
4. The Analog-to-Digital Converter (ADC) and the RF Chain needed to receiver the signal.

In this project each block was implemented as a digital circuit, while the USRP kit can be used as RF chain and ADC.

As mentioned before in the transmitter part, the main timing constraint specified by the standard is that, for 40 MHz bandwidth, the OFDM symbol's duration should be 4 µsec. in case of long GI, and 3.6 µsec. in case of short GI.

For this spec. to be satisfied, sample-domain blocks should work with a clock period of 25 ns, mainly, a 40-MHz clock, and for this stream to be continuous, some blocks will need to stop working each specific amount of time. This is discussed in detail as follows.

6.1.1 GI Removal Block:

This block's input is a continuous stream of samples. In case of long GI, we have 160 samples per OFDM symbols, but the FFT performs its operation for only 128 symbols, thus the GI should generate 128 output samples, from each 160 samples by removing the GI and then send a hold flag this flag works as another enable signal if it's '0' the block works, if it's '1' it's disable, so it's send to the FFT and the whole chain to freeze until the new OFDM symbol arrives, Note that for the FFT block to stop without losing data, all the next blocks should stop too.

6.1.2 FFT Block:

The IFFT block outputs 128 samples for 128 input samples, so it doesn't need to stop any blocks, yet its operations should be controlled by the hold signal coming from the GI block. The IFFT should work for 40 MHz clock, just like the GI block. Note that the output of the FFT is bit reversed so an additional block is added to inverse this effect.

6.1.3 Pilots Removal Block:

This block's operation is the remove the zeroes and pilots inserted in the transmitter side, as known the FFT output is 128 samples for each OFDM symbol, so the pilots removal input is 128 samples and the output is 108 samples so similarly as the GI block, this block should stop the de-mapper and all the next blocks from working until the zeros/pilots are removed, Since the block is a sample-domain block, it needs to work for 40 MHz clock as well.

6.1.4 De-mapper Block:

This is the turning point between the sample domain and the bits domain, this block have an internal clock divider as it load the data with the sample domain frequency, and the output is generated with the bit domain frequency, this clock divider is implemented using a counter which reset at a specific value depends on the type of the modulation used, as the samples domain frequency = bit frequency/ log2(M) as M is the constellation order and a clock divider is used so we don't need to stop any blocks.

6.1.5 De-interleaver Block:

The number of input of this block is equal to the number of output bits 128*NBPSC, so it's not needed to hold any blocks.

6.1.6 De-puncturing Block:

This block has the same problem as the GI and FFT it needs to generate number of bits with larger number of input bits, for example, in 3/4 de-puncturing, the output is 4 bits (needs 4 cycles) for each (3*2) input bits (needs 3 cycles), so to achieve the synchronization all the next block should be held for 1 clock cycle for each 4 input bits.

6.1.7 Viterbi Decoder Block:

This block also works in the bit domain clock frequency, and its design considerations is only to perform well when controlled by the hold signal, and it doesn’t need to stop any blocks.

6.1.8 De-scrambler Block:

Similarly, as the de-interleaver the number of input of this block equal to the number of output bits, so it's not needed to hold any blocks.

6.1.9 Data memory:

After all this holding signals the output data memory will not be serial but it will be controlled by all of the holding signals.

6.2 Components

6.2.1 Guard Removal

Guard removal is the inverse of guard insertion as explained in section 5.2.6. It removes the cyclic prefix added in the transmitter.

The block's implementation is only a hold signal controlled by a counter. Through the counter the hold signal is high when the CP samples are inputs, and it's low when the CP is done. Note that the hold signal stops all the receiver chain, so through this block the receiver would only work when the 128 samples of the OFDM symbol are at the input port.

6.2.2 FFT

The next step in decoding the signals is transforming the symbols from time domain to frequency domain through 128-point Discrete Fourier Transform operation. To implement the operation in hardware, the **Fast Fourier Transform** (FFT) algorithm is used, but since the DFT and the IDFT equations are almost identical, the IFFT block will be modified to work as an FFT. The modification is done by changing a parameter in the RTL codes that in turn does the 2 needed changes in hardware:

1. The CORDIC unit will change its operation to subtract/add instead of add/subtract as the angles' sign will be inverted.
2. The division by N will not be implemented, and signals will pass between stages without shifting.

The code is written in Verilog and parameterized to work as an FFT or an IFFT with any number of integer and fraction bits. The Signal-to-Quantization-Noise Ratio (SQNR) for the FFT reached 60 dB for 24-bit inputs: 12-bit real and 12-bit imaginary, each contains 2 bits to represent the integer part and 10 bits for the fraction part.

Note that the output of the FFT block is bit-reversed so it needs to be reordered before going into the next steps in decoding the signals in the receiver.

6.2.3 Bit-reverser

Similar to what was done in the transmitter, the FFT block's output is in bit-reversed order, so the signals need to be reordered before going into the next decoding step. At the transmitter, the GI Addition block was responsible for bit-reversing the order of the signals and adding the Cyclic Prefix, so this block's operation will be like the GI addition except that it won't add the Cyclic Prefix.

The block's operation is simple. As shown in figure 25; there will be 2 memories each of 128 entries with a write address generator and a read address generator, the write address generator generates addresses incrementing in bit-reversed order while the read address generator generates addresses that increment normally, so basically, both are 7-bit counters but the write address generator's output is reversed (0:6 instead of 6:0). The 2 memories are 128*word size and their read and write addresses are connected to the address generators.

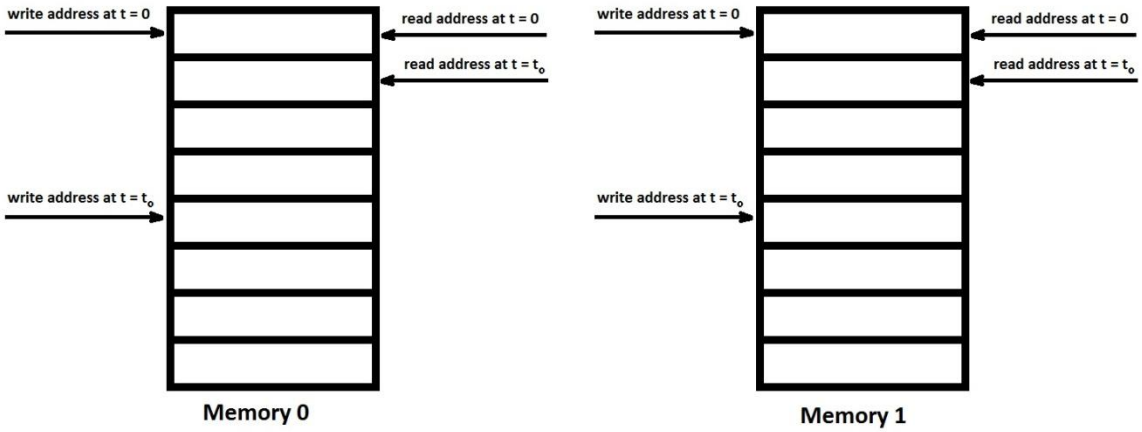

Figure 25: Bit-reverse Block Architecture

Initially, reading and writing starts at 0, but writing starts at memory 1 while reading starts from memory 0, and after 128 cycles memory 1 will be completely written so they'll swap, then the output will come out of memory 1 in order and the input will be ordered at memory 2 and they'll keep swapping each 128 cycles. There'll be an initial latency of 128 cycles, but after that all the outputs will be in order and ready to enter the pilot removal block.

The block is parameterized to work with any number of bits; and each entry in the memories hold 24 bits that represent the signal. The most significant 12 bits represent the real part, and the least significant 12 bits represent the imaginary part.

6.2.4 Pilots Removal

Pilots removal is the inverse of pilot insertion as explained in the transmitter. It removes the pilots added in the OFDM symbol in the transmitter.

6.2.4.1 Implementation:

For ensuring the pilots removal work with one symbol per clock cycle, the architecture uses two memory modules simultaneously. One module is used to write over without writing the pilots, while the other is used to read from.

6.2.5 De-mapper

In the transmitter's side each group of bits are represented in one symbol, now in the receiver side this effect should be reversed, so each symbol should be converted back to its bits. In order to do that the constellation is divided into areas each area is bounded by some values (threshold values); as shown in figure 26.

For example, considering QPSK modulation, the constellation will be divided into 4 regions as follows:

- The first region is real>0 and imag >0. Any symbol satisfying this condition can be converted to '11' bits.
- The second region is real<0 and imag >0. Any symbol satisfies this condition can be converted to '01' bits.
- The third region is real>0 and imag <0. Any symbol satisfies this condition can be converted to '10' bits.
- The fourth region is real<0 and imag <0. Any symbol satisfies this condition can be converted to '00' bits.

And so on in any type of modulation.

The implemented module includes a register and a counter, first a symbol is loaded into the register then the output is generated on the output port, and the counter counts until all the bits are generated then a new symbol is loaded.

Note that the comparison is done with the normalized threshold values.

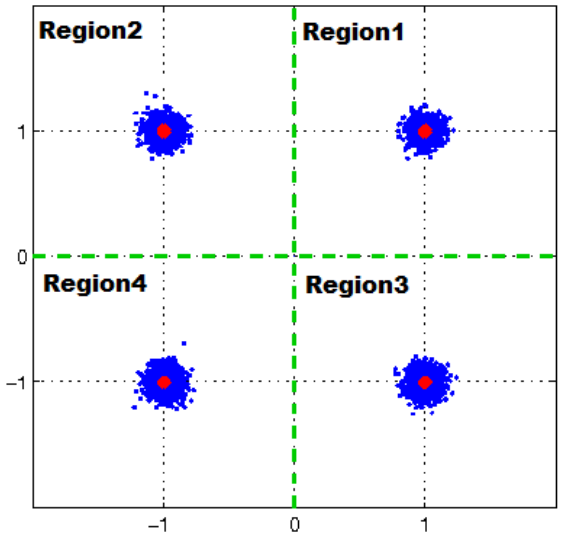

Figure 26: Constellation of QPSK at the receiver + AWGN

6.2.6 De-interleaver

De-interleaver performs the inverse relation of interleaver, is also defined by two permutations explained in equation (11), (12) & (13).

The index of the coded bit is denoted by j

$$\mathbf{i} = \mathbf{s}\ \mathbf{x}\ \lfloor j/s \rfloor + (\mathbf{j} + \left\lfloor \frac{16\ x\ j}{Ncbps} \right\rfloor)) \bmod \mathbf{s},\ \text{where } \mathbf{j} = 0, 1, \ldots\ldots, \mathbf{NCBPS} - 1 \quad (11)$$

$$\mathbf{k} = 16\ \mathbf{x}\ \mathbf{i} - (\mathbf{NCBPS} - 1)\ \mathbf{x}\ \left\lfloor \frac{16\ x\ i}{Ncbps} \right\rfloor \text{where } \mathbf{i} = 0, 1, \ldots\ldots, \mathbf{NCBPS} - 1 \quad (12)$$

$$\mathbf{s} = \max(\mathbf{NCBPS}/2, 1) \quad (13)$$

6.2.6.1 Implementation

For ensuring the de-interleaver works with one bit per clock cycle, the architecture uses two memory modules simultaneously. One module is used to write over, while the other is used to read from. There is a third memory which store the address generated by the equations 11 and 12 so that the written data is written with this address and is read in order with a simple counter.

6.2.7 De-puncturing

In the receiver's side the puncturing process done at the transmitter side is inversed, so zeros are inserted at the locations of the punctured bits before entering

the Viterbi decoder, and the output is generated as 2 parallel bits instead of 1-bit stream to be compatible with the decoder input.

The implemented module includes two registers, the first one is loaded from the input until it's full, then it's copied to the second register with the required scheme (inserting zeros in the required locations), then output is loaded from the second register as two parallel bits, the critical issue here is that the input is loaded in more cycles than the output is generated in. For example, in R=3/4 input bits loading needs 4 cycles and the output is generated in 3 cycles so the same idea of the puncturing block is used, which was a holding flag that disables all the previous blocks, but here all the next blocks are held (disabled), so now we have 4 cycles to load the input and 3 cycles to generated the output + 1 hold cycle.

The following figures 27 and 28 show how the de-puncturing is done.

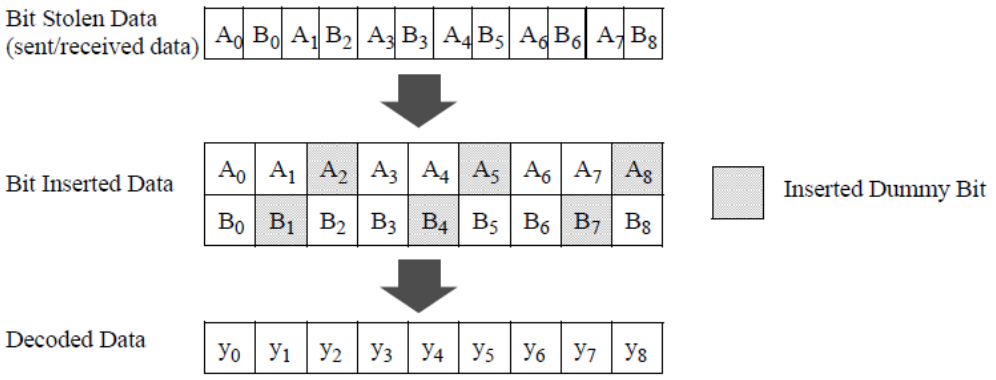

Figure 27: 3/4 rate de-puncturing process

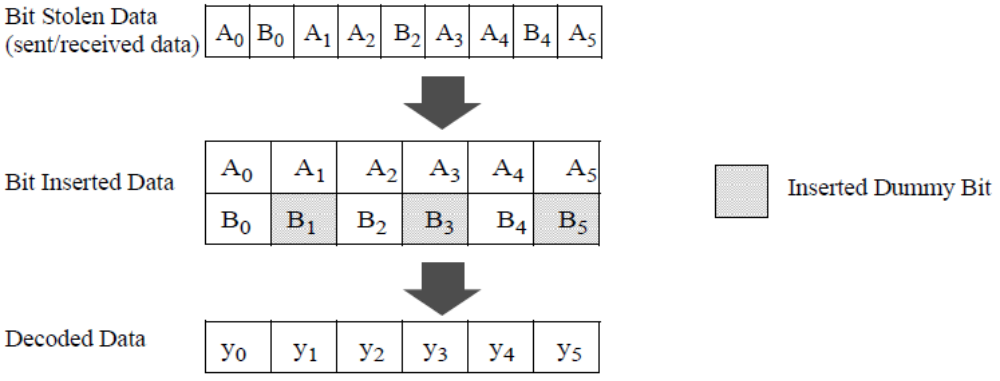

Figure 28: 3/4 rate de-puncturing process

6.2.8 Viterbi Decoder

Viterbi Decoders are employed in digital wireless communication systems to decode the convolutional codes which are the forward error correcting codes. Although widely-used, the most popular communications decoding algorithm, the Viterbi Algorithm (VA), requires an exponential increase in hardware complexity to achieve greater decoding accuracy. When the applications based on wireless technology were developed tremendously with the world, the constraint length associated with the input bits increased, hence the larger constraint length needs to be implemented with less hardware and less computations for decoding the original data [14].

In this section, the concept of Viterbi decoder is discussed, and how it decodes the data and overcomes impulsive errors in the transmitted bit stream, then area optimization is discussed for the memories needed in the design, and then finally the results are presented on virtex-7 board, i.e. the utilization, maximum frequency, latency and maximum throughput.

The concept of the Viterbi algorithm is based on the fact that the convolutional encoder at the transmitter is a state machine which goes from a state to another according to the input bit stream of the message. Thus, for a known initial state (zero state) a given input stream has a known sequence of state transitions and output stream. The receiver surely knows how the encoder can go from any state to another by which input and how the output will be, this is known as the encoder's constellation.

The used encoder for IEEE 802.11n WLAN standard has one input and two outputs, thus have an encoding rate of ½. It also consists of a 6-bit shift register; thus it has 64 states (from state 0 to state 63). Since the encoder and decoder of this size will be extremely large and hard to explain, we'll consider a simple encoder and decoder to discuss the operation and the functional blocks for the digital implementation.

Figure 29 shows a simple encoder with 2-bit shift register, i.e. 4 different states. Figure 30 shows the corresponding constellation for the encoder, this should be known by the receiver to effectively decode the data. As discussed previously, the encoder is a simple state machine with a known state diagram.

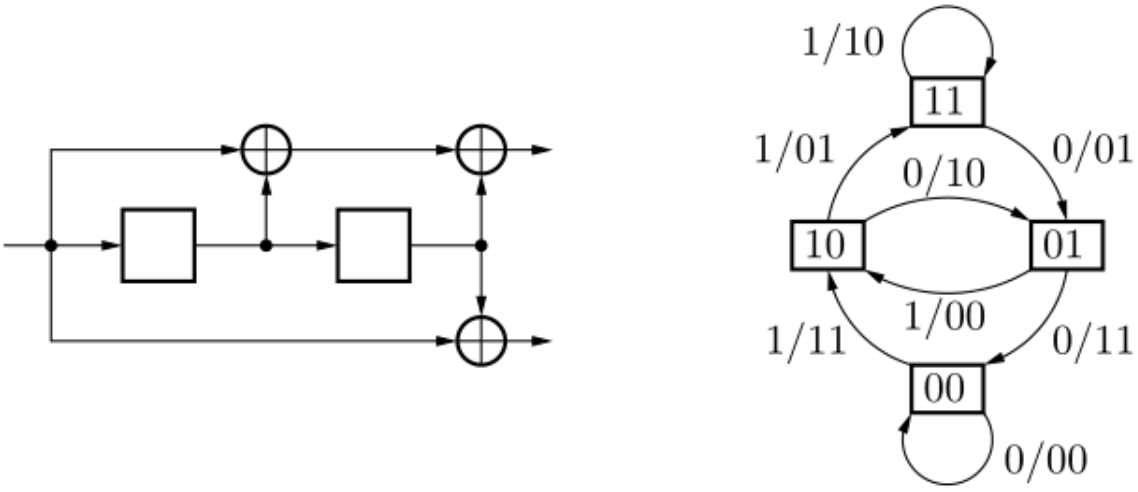

Figure 29 : Simple encoder and its state diagram

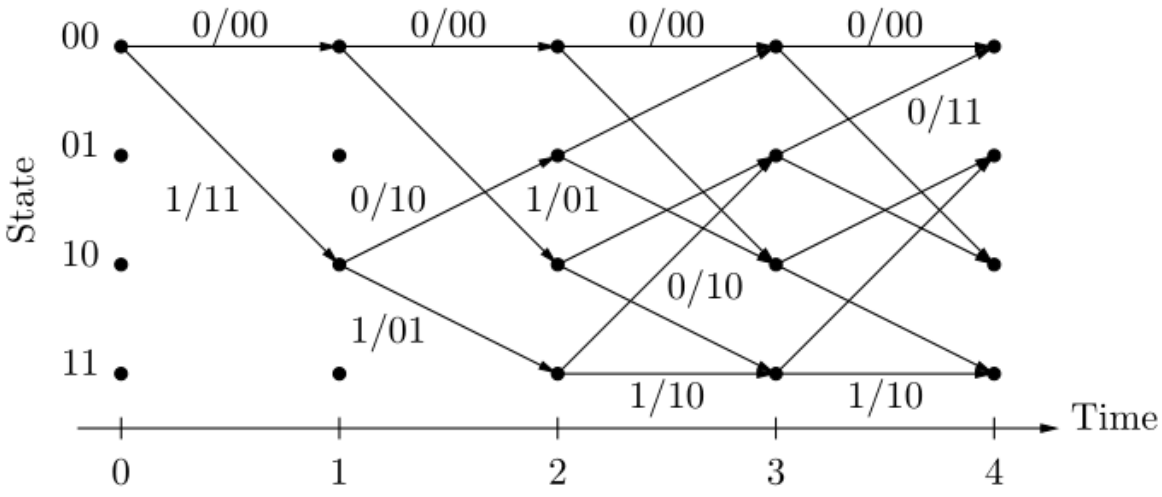

Figure 30: Encoder's Constellation

The concept of Viterbi decoding is based on legal and illegal state transitions, that is, a state machine with zero initial state has legal transitions at each state. For example, as shown, in the constellation in figure 30; state 0 has only two possible next states which are state 0 and state 2, thus, if the decoder received an input code stream which corresponds to an illegal transition from state 0 to state 1, it will detect the error. Moreover, it can also correct this error, by keeping track of the other codes which correspond to other legal transitions for an impulsive error, thus the decoder will have a cracked path (not totally 100% legal), and using path metric algorithm, it will look for the nearest legal path (sequence of legal transitions, each transition has a corresponding input message bit and output 2-bit code word) and extract the message of this legal path.

The state path algorithm can be explained by the following example: first, the decoder assumes 4 available paths; each path ends with a state of the 4 states of our example. The final task is to extract the winning path which corresponds to the decoded and extracted message. Each path of these 4 paths has a path metric, which a representative number is used to show how legal this path is. The starting point of all paths is known which is state 0. The decoder takes the first code word and calculates the metric of this word with all available code words (00 01 10 11), and updates all path metrics using their branch metrics.

Note that each path has only two available code words, since the constellation has only two incoming arrows (state transition) with each arrow having an output code word. For example, if the first code word is 00, the first path metric will increment since 00 is exactly the same as the output when the encoder has a transition from state 0 to state 0, while the third path metric will not be incremented, since the legal output for a legal transition from state 0 to state 2 is 11 and this is completely different from 00.

The process of message extraction is done by 4 main steps. The first step is done while the metrics are updated (incremented for winning case or kept at its old value for losing case), in this case the corresponding bit of each code word is written in a memory in each cycle, hence, 4 parallel bits are written at each decoding cycle. The second step is to find the winning path, which is the path of the maximum path metric. The third step is read the bits of this winning path, however those bits are read in reverse order, since the reading and writing operations have different directions, so one last step is need, which is step 4. It is a basic LIFO stack used to flip the extracted winning message of the winning path. A simple waveform showing the main stages for the Viterbi decoding is shown in figure 31.

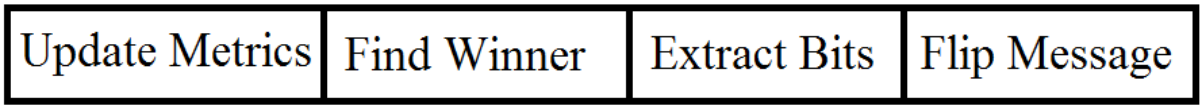

Figure 31: Basic stages for the Viterbi decoder

The state metric mechanism needs to be reduced to functional blocks to be ready for implementation. The first block is the branch metric unit (BMU). The BMU for hard decision decoding is used to calculate the number of similar bits between the incoming code word and a given legal word. This can be simply done using XNOR gates and half adders. The block diagram of the branch metric unit is shown in figure 32.

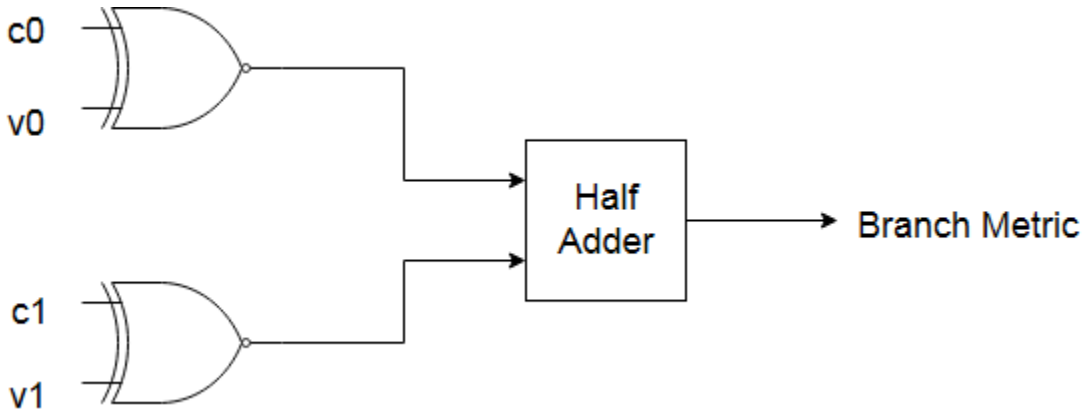

Figure 32: Block diagram for the branch metric unit

The second block is based on the BMU which is the Add-Compare-Select (ACS), it's used to update the path metric for each path of the available paths (2 raised to the power of the constraint length, which equals to 4 in this example and 64 in the project for IEEE 802.11n PHY standard). This block is mainly a register to hold the metric value and adders to add the old value with the incoming branch metric from the BMU. However, since each point on the constellation has two incoming arrows (two possible state transitions), this block need to be doubled, i.e. we have two adders for two incoming path metrics and two branch metrics, then select only one value is selected (larger value) and the other one is discarded, since it's less likely to be legal path. The block diagram for the ACS unit is shown in figure 33.

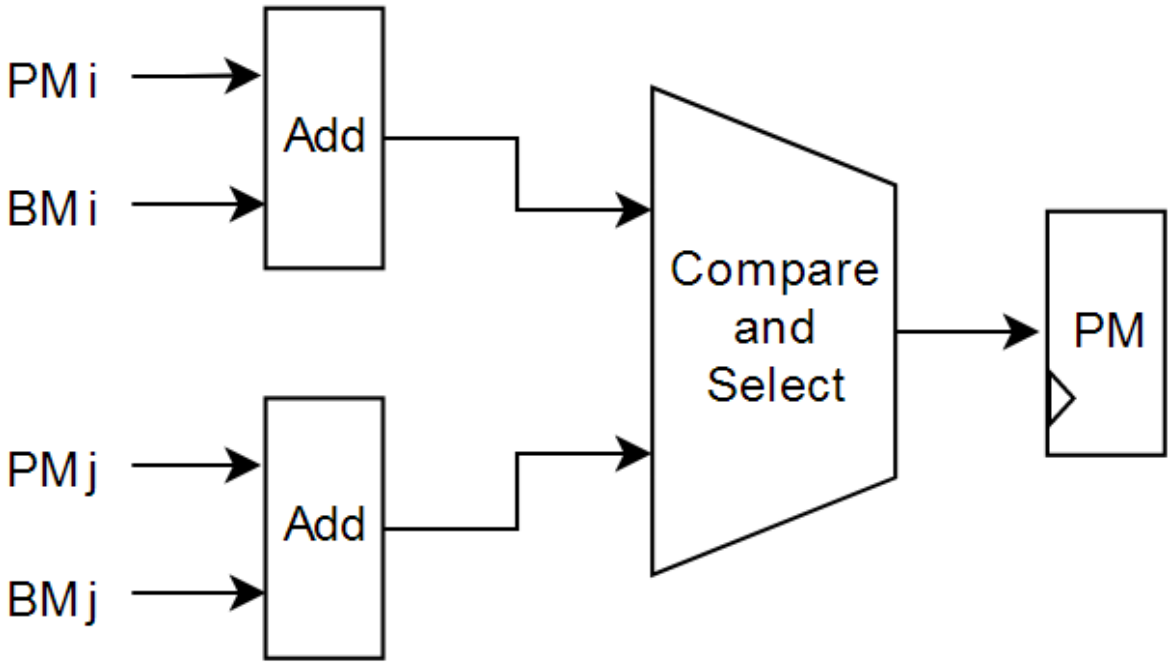

Figure 33: Block diagram for the ACS unit

The third block is the memory, called as the trace back memory or the trace back unit (TBU). This memory is simply a holder for the corresponding messages for each path of the available paths. It's generated and filled during the first stage by the ACS units. Each ACS unit writes 1 or 0 in the corresponding place in the memory for each cycle. For example, if all ACS units have chosen the first path after the comparison, and those paths had corresponding input (pre-known from the trellis diagram), then the first column of this memory should be zeros. The block diagram with sizes depicted as shown in figure 34.

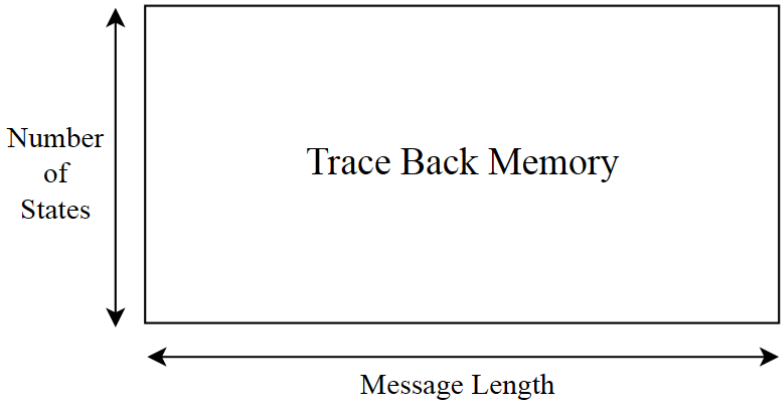

Figure 34: The Trace Back Memory

Stage 1 of decoding is used to write in the TBU the bits of all possible messages, while stage 3 is used to extract the winning message from the end to the start (reverse order), thus, a stage is needed to efficiently get the address of the winning path (the address range is from 000000 to 111111 for K=7 encoder as the case in the project). The task of finding the winning path is done by a unit called the Survival Metric Unit (SMU). This unit consists of a set of comparators used to find the maximum metric; the organization of that comparator is a tree-like hierarchical structure. However, it needs some modifications to find the 6-bit address of this metric, not only the maximum value, but also the index of this value. To do this, the 64-path metrics are divided into 32 pairs and the maximum of each pair is found, then this maximum is forwarded with a selection bit (0 if the upper metric is select and 1 otherwise), the 32 forwarded metrics are also divided into 16 pairs to do the same thing, and hence we have $\log_2(64) = 6$ pipelined stages for the SMU. The block diagram for this stage is shown in figure 35 (for a simple case of 4 possible paths only); the operation of each comparison unit is like the ACS.

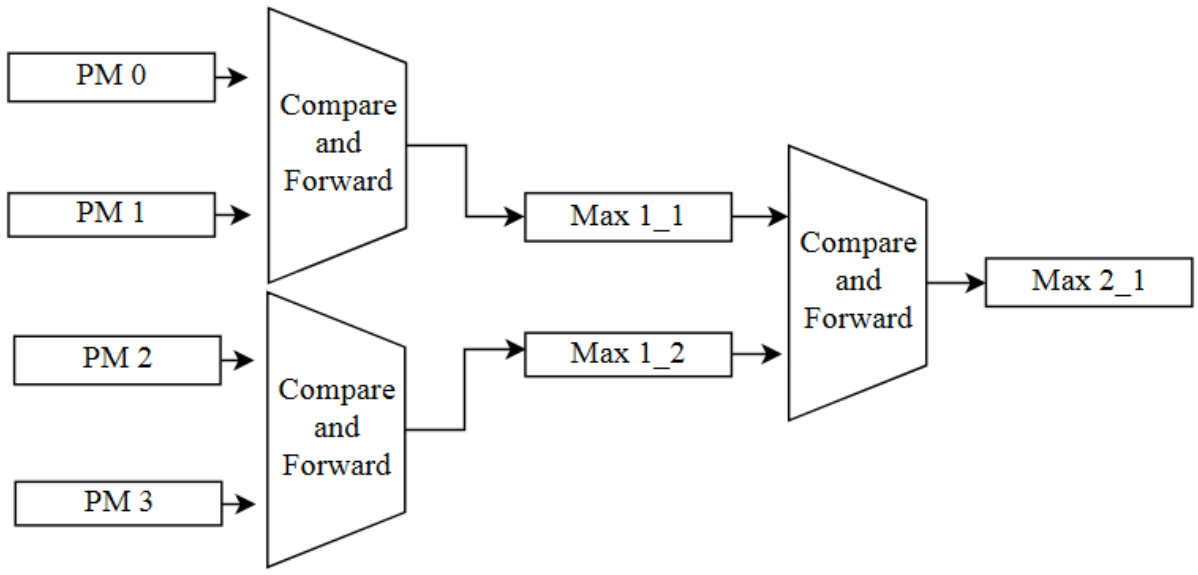

Figure 35: Block diagram for simple SMU stage

The third stage, as discussed before, is a control unit used to read the message in reverse order, since the SMU outputs the address of the winning path, which points at the last part of the message. The trace back technique catches the address of the winning path and reads the corresponding bit, then generates the predecessor address according to the decision bit, since each state has only two possible predecessors, we can calculate them using only single bit, and not by

storing the whole address in the TBU memory, this technique has reduced the memory size to 1/6 of its conventional size, and for sure it enhanced the maximum clock speed for the design.

While the third stage is extracting the message bits in reverse order, it writes in the LIFO stack, which is read from in the fourth and last stage, to finally output the message in correct order. The length of the LIFO buffer equals to the message length, which equals to the number of coded bits per symbol multiplied by number of data sub-carriers multiplied by the coding rate.

A general block diagram for the Viterbi decoder is shown in figure 36; it shows a simple example with 8 states with message length of 10. Stage 1 fills the memory; stage 2 finds the address of the last bit; stage three traces back the winning message; and finally stage 4 reorders the message.

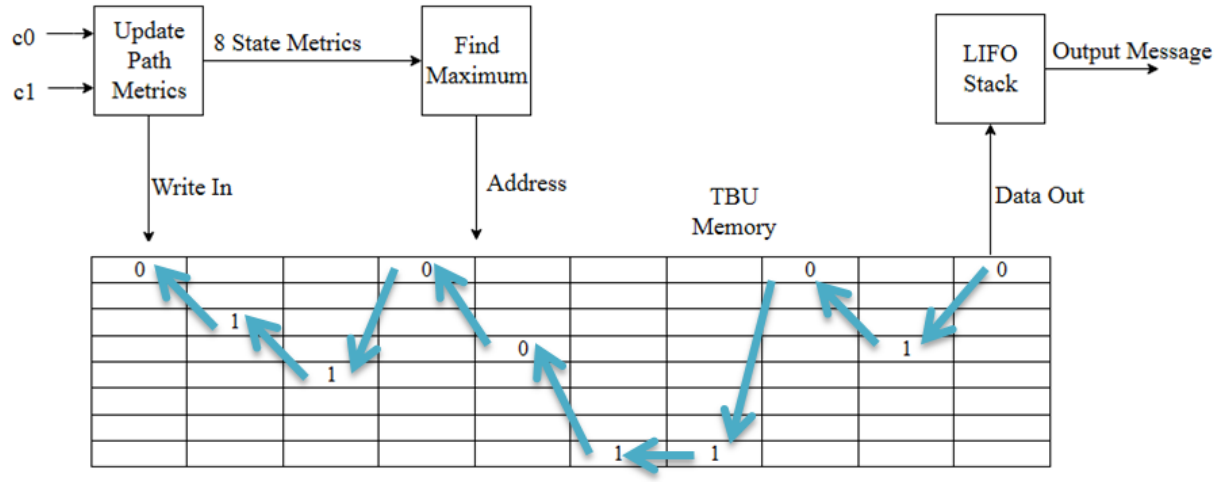

Figure 36: Viterbi detailed block diagram and trace back technique

The last step is to pipeline the design. The decoder needs to keep receiving codes and keep outputting corresponding messages without the need to wait for initial latency multiple times. The waveform for the pipeline is shown in figure 37; it shows that the design needs some replication. First, since stage 1 is being called multiple times in sequence while stage 3 of the first message is not done yet, three TBU memories are needed to handle this problem. Second, for stage 4, only two LIFO stacks are needed, not three; since three pipelined messages can be flipped without stalling using only two buffers.

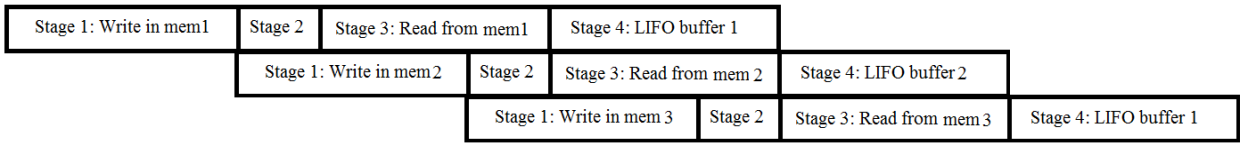

Figure 37: Waveform for pipelined 4-stage Viterbi decoder

Simulation Results

In this section, the utilization and clock constraints results are shown on Vivado tool version 2016.4 for Virtex-7 xc7vx485t-3ffg1157 board. The timing results are shown in figure 38, while the table of utilization is shown in figure 39. The design was synthesized and implemented, and the results show that the minimum clock period is 4ns; which gives a maximum achievable frequency of 250-MHz. The total utilization is below 1% of the virtex-7 board.

		Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1	Yes	TS_clk = PERIOD TIMEGRP "clk" 4 ns HIGH 50%	SETUP HOLD	0.022ns 0.003ns	3.978ns	0 0	0 0

Figure 38: Timing results for the Viterbi decoder

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	2,997	607,200	1%
Number used as Flip Flops	2,928		
Number used as Latches	69		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	9,037	303,600	2%
Number used as logic	5,743	303,600	1%
Number using O6 output only	2,822		
Number using O5 output only	54		
Number using O5 and O6	2,867		
Number used as ROM	0		
Number used as Memory	3,288	130,800	2%
Number used as Dual Port RAM	3,288		
Number using O6 output only	3,084		
Number using O5 output only	36		
Number using O5 and O6	168		
Number used as Single Port RAM	0		
Number used as Shift Register	0		
Number used exclusively as route-thrus	6		
Number with same-slice register load	0		
Number with same-slice carry load	6		
Number with other load	0		
Number of occupied Slices	2,582	75,900	3%
Number of LUT Flip Flop pairs used	9,111		
Number with an unused Flip Flop	7,051	9,111	77%
Number with an unused LUT	74	9,111	1%
Number of fully used LUT-FF pairs	1,986	9,111	21%
Number of unique control sets	56		
Number of slice register sites lost to control set restrictions	139	607,200	1%

Figure 39: Total utilization table for the Viterbi decoder

6.2.9 De-scrambler

All the data bits received by 802.11a are descrambled after decoding it, using the same frame synchronous 127 bits sequence generator used in scrambler as explained in section 4.2.1. The de-scrambler uses the generator polynomial S(x) as follows in equation 14.

$$S(x) = x^7 + x^4 + 1 \quad (14)$$

7. Synchronization

7.1 Introduction

Synchronization is an essential task for any digital communication system. Without accurate synchronization algorithms, it is not possible to reliably receive the transmitted data. The first task of synchronization is the packet synchronization, which is the ability to detect the arrival of the packet at the receiver and make sure that this packet belongs to the desired wireless standard. The second task is the symbol synchronization which means to indicate the start of incoming packets without prior knowledge. The third task is the frequency synchronization by finding the frequency mismatch between the oscillators of the transmitter and the receiver [15].

7.2 Packet Synchronization

Packet detection is the task of finding an approximate estimate of the start of the preamble of an incoming data packet. The simplest algorithm is to measure the received signal energy. When there is no packet being received, the received signal consists only of noise. When the packet starts, the received energy is increased by the signal component, thus the packet can be detected as a change in the received energy level. The received signal energy accumulated over some window of length L to reduce sensitivity to large individual noise samples. A better alternative algorithm is called the double sliding window packet detection. This algorithm calculates two consecutive sliding windows of the received energy. The basic principle is to form the decision variable as a ratio of the total energy contained inside the two windows.

The previous two algorithms have common disadvantages. For example, the value of the threshold which could be used to decide when an incoming packet starts depends on the received signal energy. When a desired packet is incoming, its received signal strength depends on the power setting of the transmitter and on the total path loss from the transmitter to the receiver. All these factors make it quite difficult to set a fixed threshold.

Another disadvantage is that the detected packet could be from any source and not specified for the desired wireless standard. To solve these problems the receiver should use the known structure of the preamble to increase the performance of the packet synchronization.

As shown in figure 40 the structure of the IEEE 802.11n frame starts with 10 periods of short training symbols. From that, another effective method can be used, which resembles the double sliding window algorithm but it takes advantage of the periodicity of the short training symbols at the start of the preamble. This approach is called the delay and correlate algorithm.

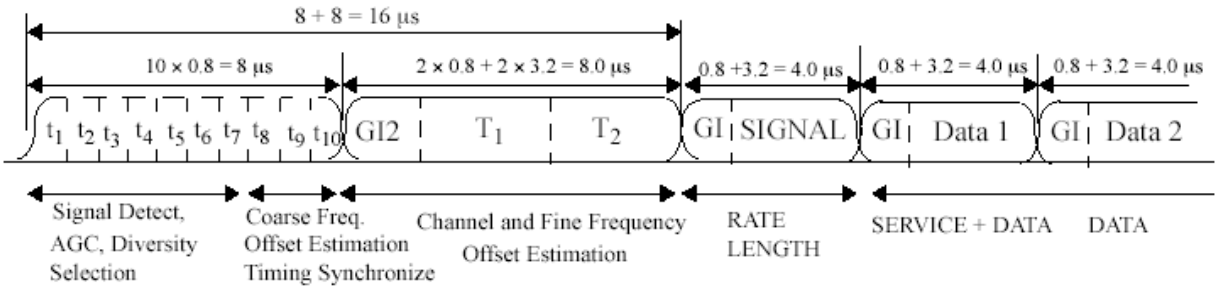

Figure 40: IEEE 802.11n frame format

Figure 41 shows the signal flow structure of the delay and correlate algorithm. The Figure shows two sliding windows C and P.

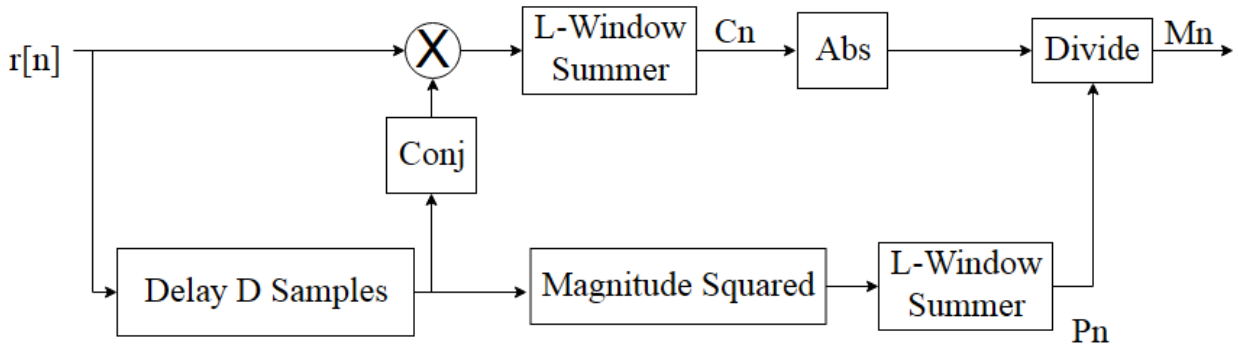

Figure 41: Block diagram for the packet detection algorithm

The C window which is described in equation 15 is a cross correlation between the received signal and the complex conjugate of a delayed version of the received signal accumulated over some window of length L to reduce sensitivity to large individual noise samples, this delay (D) is equal to the period of the short training symbols.

$$C_n = \sum_{k=0}^{L-1} r[n+k]r[n+k+D]^* \qquad (15)$$

The P window described in equation 16 calculates the received signal energy during the cross-correlation window. The value of the *P* window is used to normalize the decision statistic, so that it is not dependent on absolute received power level.

$$P_n = \sum_{k=0}^{L-1} r[n+k+D]r[n+k+D]^* = \sum_{k=0}^{L-1} |r[n+k+D]|^2 \qquad (16)$$

Then the decision statistic is calculated from equation 17 and compared to the threshold which is less than one because of the power normalization.

$$M_n = \frac{|C_n|}{P_n} \qquad (17)$$

Figure 42 shows an example of the decision statistic M_n for IEEE 802.11n preamble in 10dB SNR. When the received signal consists of only noise, the output C_n of the delayed cross correlation is zero-mean random variable, since the cross correlation of noise samples is zero. This explains the low level of M_n before the start of the packet. Once the start of the packet is received, C_n is a cross correlation of the identical short training symbols, which causes M_n to jump quickly to its maximum value; this jump gives a quite good estimate of the start of the packet.

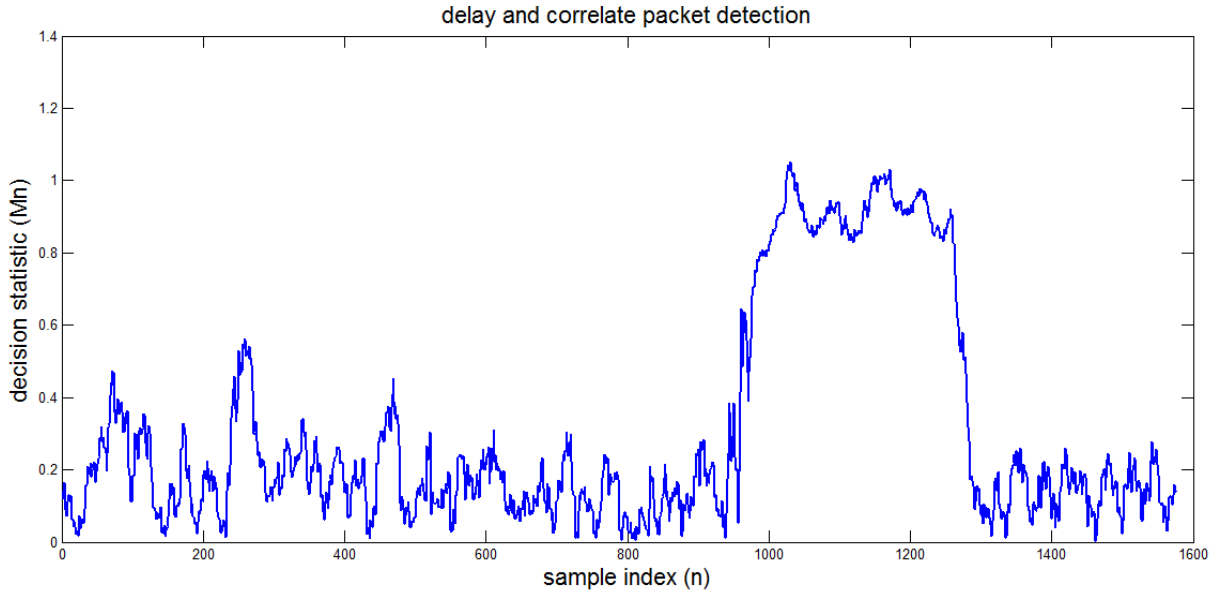

Figure 42: Delay and Correlate MATLAB Output

To simplify the hardware implementation some changes happened. First, it's easy to get rid of the division by comparing the $|C_n|$ with the multiplaction of P_n to the threshold. Second, to calculate the magnitude of the C_n a vectoring CORDIC which is described in chapter 5 is used to simplify the implementation.

7.3 Symbol Synchronization

Symbol Synchronization refers to the task of finding the precise moment of when individual OFDM symbols start and end. After the packet detector has provided an estimate of the start edge of the packet, the symbol timing algorithm refines the estimate to sample level precision. The refinement is performed by calculating the cross correlation of the received signal and a known reference as described in equation 18; where t_k is the known reference.

$$\hat{t_s} = \max index\ n\ \left|\sum_{k=0}^{L-1} r[n+k]\, t_k^*\right| \quad (18)$$

The value of *n* that corresponds to maximum absolute value of the cross correlation is the symbol timing estimate. The length of the cross correlation determines the performance of the algorithm. Larger values improve performance, but also increase the amount of computation required. The hardware implementation is shown in figure 43.

It is possible to use only the sign of the reference and received signals, effectively quantizing them to one-bit accuracy. This greatly simplifies the hardware implementation, since instead of actual multiplications only XNOR gates are used then compare the summation of the XNOR outputs (decision statistic) with a threshold to detect the correct symbol timing point.

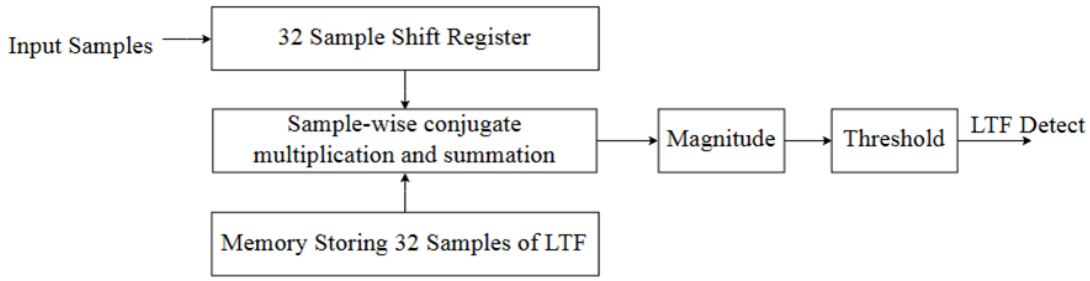

Figure 43: Timing offset estimation technique using cross correlation

The problem with the sign-based technique is the sensitivity to the mismatch of the oscillator frequency. Hence, it's better to use the cross-correlation technique even if it's more complex and utilized more resource.

Figure 44 shows the output of the cross-correlator that uses the first 32 samples of the long training field as a reference signal. The simulation was run in Additive White Gaussian Noise channel with 3dB SNR. The high peak clearly shows the correct symbol timing point.

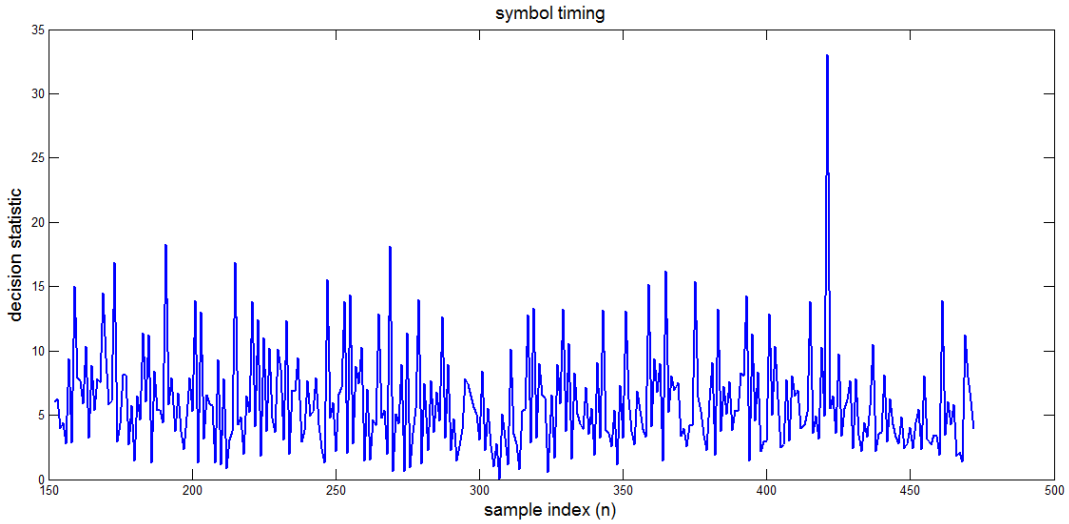

Figure 44: Correlation based timing estimation algorithm MATLAB output

7.4 Frequency Offset Estimation and Compensation

Due to the mismatch in oscillator frequencies of both radio frequencies (RF) transmitter and receiver, the carrier frequency is not the same for both sides. The block diagram for the up-conversion and down-conversion with mismatch in frequencies is shown in figure 45; it shows that a residual phase error is generated at the output, which affects the base-band later.

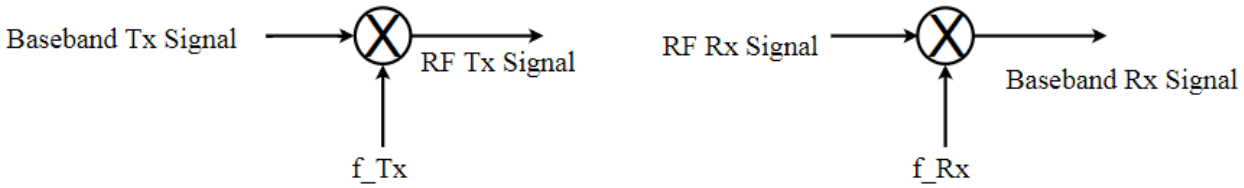

Figure 45: Block diagram for the up and down conversion process

For a carrier frequency f_c, both f_{Tx} and f_{Rx} are not equal to the carrier frequency exactly, equations 19 and 20 shows the problem of this frequency offset.

$$\boldsymbol{RF_{Tx} = BB_{Tx} e^{-j2\pi f_{Tx} t}} \quad (19)$$

$$\boldsymbol{BB_{Rx} = RF_{Rx} e^{-j2\pi f_{Rx} t} = BB_{Tx}\, e^{-j2\pi \Delta f t}} \quad (20)$$

The residual frequency Δf contributes a phase which keeps increasing with time, i.e. each sample at index n will have a phase error of $2\pi\Delta f T_s$ with respect to the previous sample; this causes rotation of the sample in both frequency and time domains in the polar form. For example, if the transmitted symbols are BPSK symbols for stream bits of ones resembled as 1 in the real axis, it will be received as bunch of complex signals of magnitude one and a rotating angle moves on a circle; so after some number of samples, the phase error will exceed pi and the steam bits will be received as zeros then ones and so on.

A data aided approach for the estimation of frequency offset in the time-domain will be discussed and implemented. The idea is that the preamble is known for the receiver, the preamble consists of a short training field (STF) of 10 periods, followed by a long training field (LTF) of 2 periods with a guard interval as discussed in the standard specifications. The periodic data could be correlated with a delayed version of it with a delay equals to its period, ideally, it should give the

squared magnitude of the data. However, the additional phase error due to the frequency offset will be added to this magnitude, thus we could calculate the phase of this term using rotational CORDIC, as explained in section 5.2.5. Relations showing the correlation and calculation of offset are shown in equations 21 and 22; where r[n] is the received signal at sample index n, s[n] is the ideal value of the known data and T_s is a sampling rate of 25ns for the high throughput mode with 40-MHz Channel BW.

$$\boldsymbol{z = \sum_L r[n] r[n+D]^* = e^{-j2\pi\Delta f D T_s} \sum_L |S[n]|^2} \quad (21)$$

$$\boldsymbol{\Delta f = \frac{-phase(Z)}{2\pi D T_s}} \quad (22)$$

A block diagram for the frequency offset estimation is shown in figure 46; the only tunable parameter in the design is the summation window length L, as the length increase, the initial latency and memory resources increases while the performance degradation by the noise decreases.

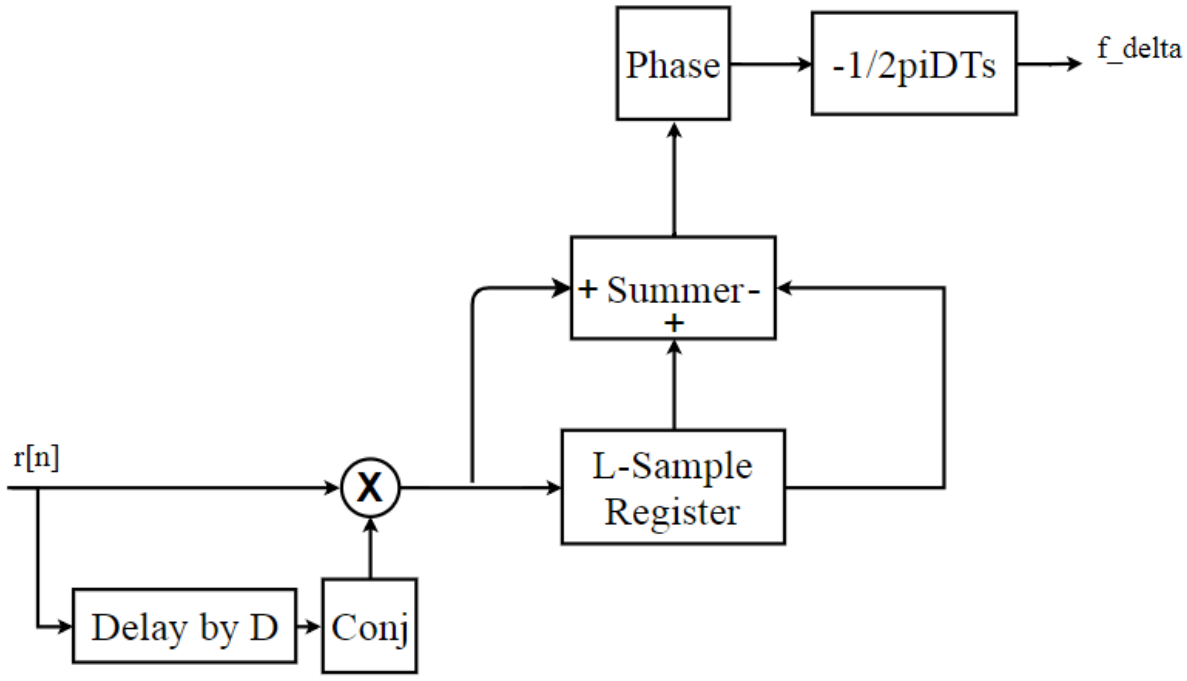

Figure 46: Frequency offset estimation using pre-known data r[n]

Based on the value of the period D, the range of offset frequencies that could be estimated is known. As discussed in the standard specifications chapter; the short training field has period of 32 samples in the 40-MHz mode, that have

maximum estimated offset of 625 KHz, and for the long training field of 128 samples period, the maximum offset is 156.25 KHz.

The estimation of frequency is done on two steps, coarse frequency estimation is done on the STF for its long range of estimated frequencies, followed by fine frequency estimation on the LTF, knowing that the fine frequency estimation is done after the coarse frequency compensation; i.e. rotating the samples in the opposite direction using the estimated frequency and a rotational CORDIC. A simple diagram showing the operation is shown in figure 47.

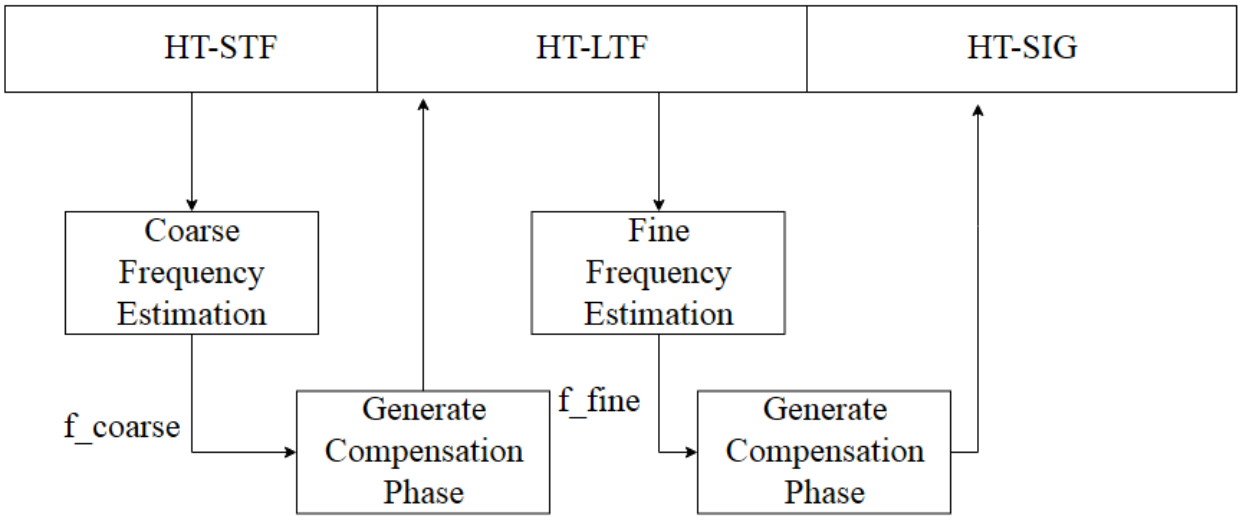

Figure 47: Coarse and fine frequency estimation and compensation

As shown in the figure, the initial coarse estimate is done on the STF, and then this value of frequency is used to generate a rotating angle in the opposite direction to compensate the LTF. Then the coarse compensated LTF is used for the fine frequency estimation, and again the value of the fine offset is used to generate a second rotating angle in the inverse direction to compensate the next fields in the frame, which are the HT-SIG field followed by DATA field.

7.5 Synchronization Top Block Diagram and Utilization

In this section, the full operation of the synchronization blocks is discussed starting from detecting the packet ending with correct timing and fine compensated I and Q baseband signals to be forwarded to the receiver. The block diagram is shown in figure 48. The original I and Q signals coming from the 12-bit ADCs are fed through the input ports of the blocks. First, the packet detection algorithms work on those signals and detect the IEEE 802.11n, preamble then raises the Detect flag. Second, coarse frequency estimation is done on the STF periods and the timing estimation algorithm waits for the LTF for the sign similarity algorithm to raise the LTF_{Detect} flag. Note that the packet detection can have a delay and raises the Detect flag after 3 to 5 periods of the STF. However, we have 10 periods of the short training fields, so the coarse frequency estimation can still have at least 5 complete periods to estimate the frequency offset. The generated coarse frequency is used to compensate the LTF with the proper initial phase when the timing offset is estimated. When the LTF starts, the fine frequency estimation begins to estimate the offset when the coarse compensation block outputs its first coarse compensated sample. Finally, the fine frequency is used to compensate the other fields and feed the baseband receiver with a fine compensated I and Q signals called I_{fine} and Q_{fine} in the block diagram.

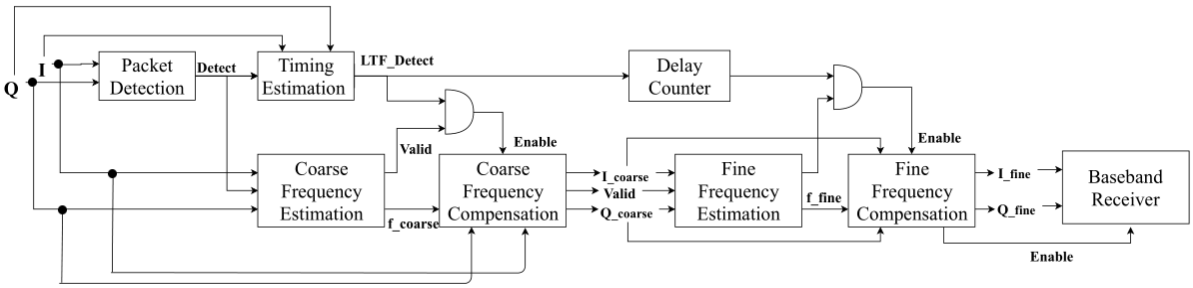

Figure 48: Top block diagram for the synchronization blocks

8. Results

8.1 MATLAB Results

After reading the standard, a high level MATLAB model was done for the transmitter and receiver with different MCS Indices. The obtained BER curves are shown in figures (49) and (50).

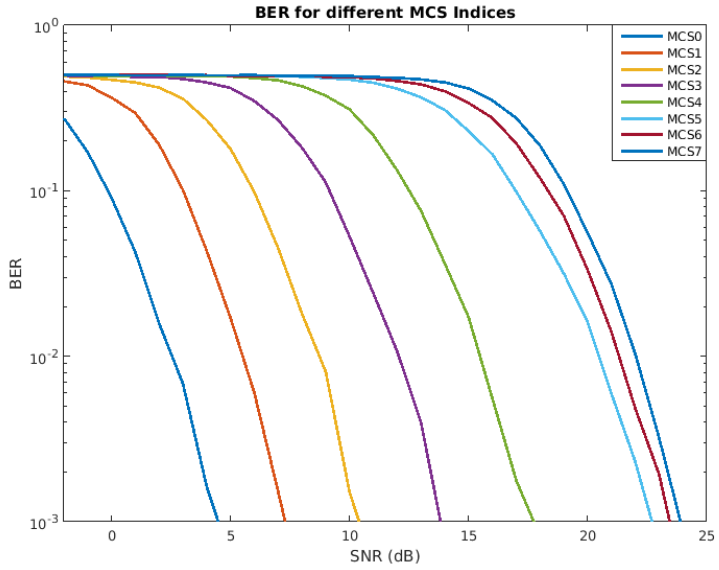

Figure 49: BER curves for long GI

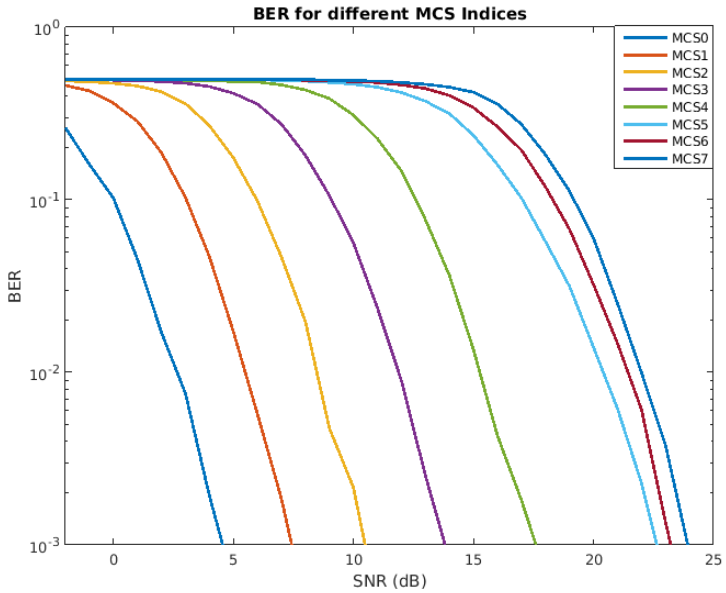

Figure 50: BER curves for long GI

From the figures it can be noticed that the BER is almost identical for short and long GI, and this is reasonable since the simulated channel is only AWGN channel with no fading. The channel model doesn't have fading since the channel is only a few centimeters between 2 antennas, and the GI is only inserted to prevent ISI and ICI which occur as a result of fading.

8.2 FPGA Results

8.2.1 Transmitter

Figure 51 shows Utilization on ZYNQ ZC702 Evaluation Board:

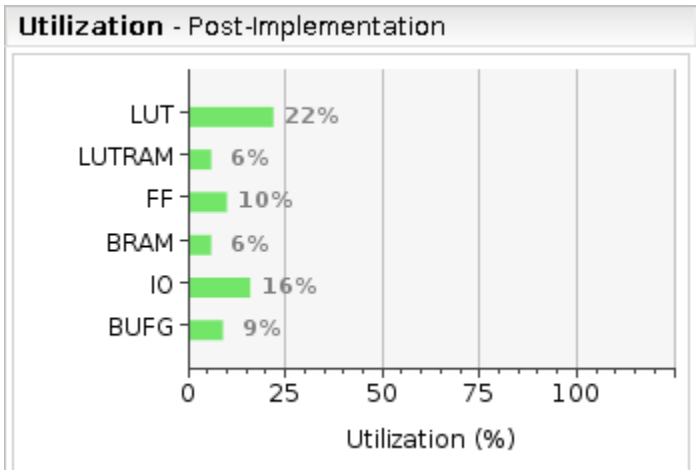

Figure 51: Transmitter's Utilization on ZYNQ Board

Figure 52 shows Utilization on Ultrascale xcvu440-flga2892 FPGA part number:

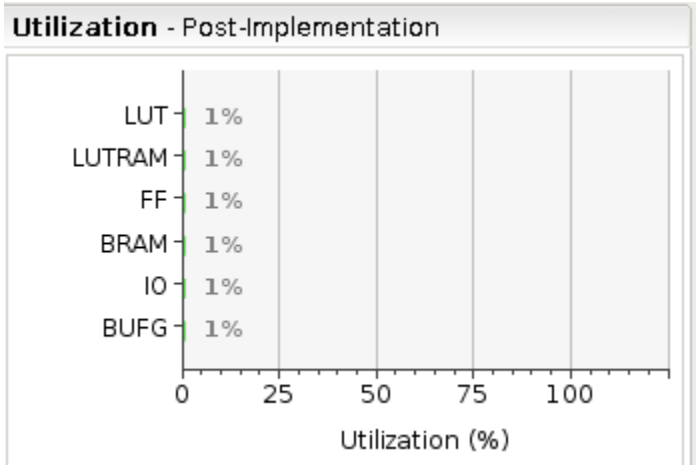

Figure 52: Transmitter's Utilization on Ultrascale FPGA

Figure 53 shows the Design's Timing Summary on ZYNQ board:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.370 ns	Worst Hold Slack (WHS):	0.015 ns	Worst Pulse Width Slack (WPWS):	1.875 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	25059	Total Number of Endpoints:	25059	Total Number of Endpoints:	11976

All user specified timing constraints are met.

Figure 53: Timing Summary for the transmitter on ZYNQ board

8.2.2 Receiver

Figure 54 shows Utilization on ZYNQ ZC702 Evaluation Board:

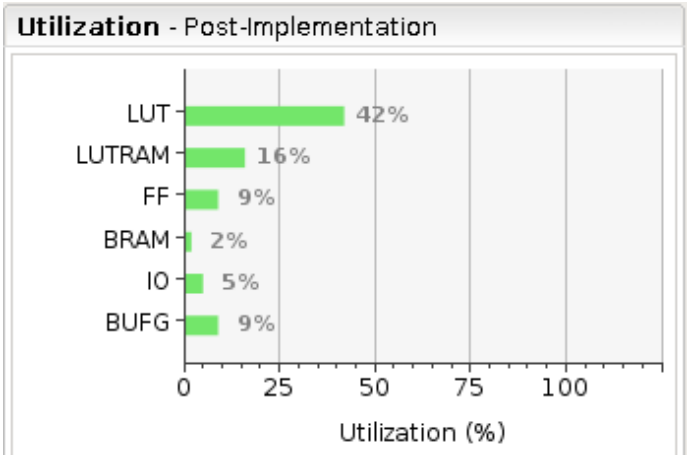

Figure 54: Receiver's Utilization on ZYNQ Board

Figure 55 shows Utilization on Ultrascale xcvu440-flga2892 FPGA part number:

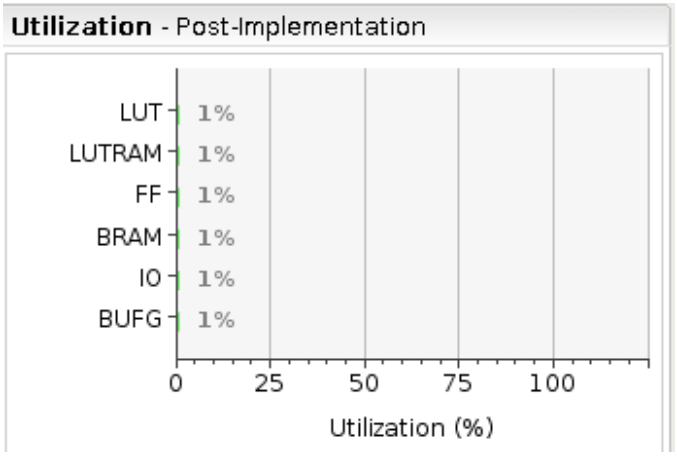

Figure 55: Receiver's Utilization on Ultrascale FPGA

Figure 56 shows the Design's Timing Summary on ZYNQ board:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.080 ns	Worst Hold Slack (WHS):	0.029 ns	Worst Pulse Width Slack (WPWS):	1.875 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	28244	Total Number of Endpoints:	28244	Total Number of Endpoints:	13667

All user specified timing constraints are met.

Figure 56: Timing Summary for the receiver on ZYNQ board

8.2.3 Synchronization

Figure 57 shows Utilization on ZYNQ ZC702 Evaluation Board:

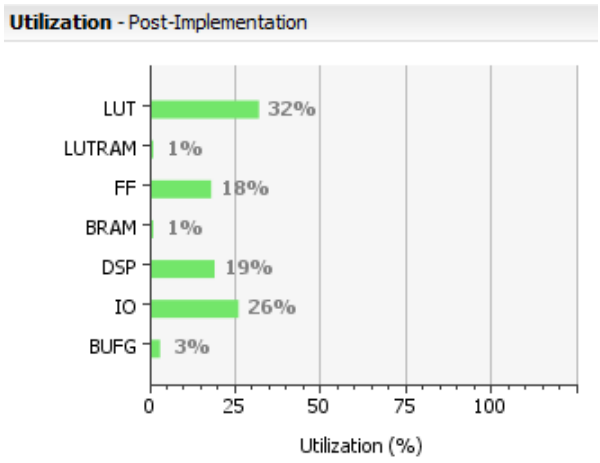

Figure 57: Synchronization's Utilization on ZYNQFPGA

Figure 58 shows Utilization on Ultrascale xcvu440-flga2892 FPGA part number:

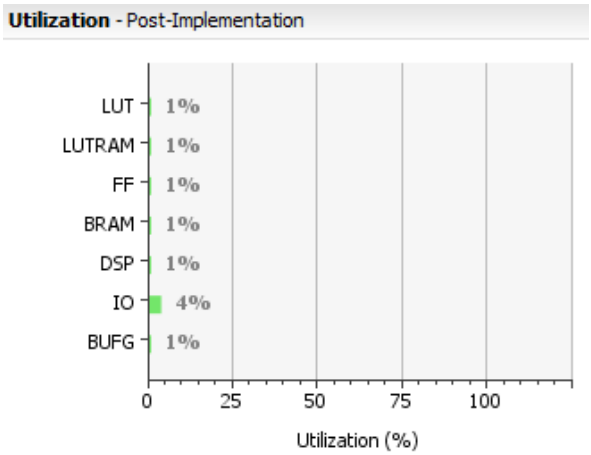

Figure 58: Synchronization's Utilization on Ultrascale FPGA

Figure 59 shows the Design's Timing Summary on ZYNQ:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.370 ns	Worst Hold Slack (WHS):	0.015 ns	Worst Pulse Width Slack (WPWS):	1.875 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	25059	Total Number of Endpoints:	25059	Total Number of Endpoints:	11976

All user specified timing constraints are met.

Figure 59: Timing summary of synchronization

As shown in the figures, the designs meet the constraints with no negative slack, and their utilization doesn't exceed 1% on Ultrascale, so the project can be used in testing designs that are partitioned on Ultrascale FPGAs.

9. Future Work

9.1 Channel Estimation

The channel estimation algorithms were investigated, and they were mainly divided into 2 main methods.

9.1.1 LTF-based Estimation

It's simply based on monitoring the channel response through comparing the original LTF values with the received ones. The estimation can be done in frequency or time domain, and it gives only an initial estimate that fails when channel variations occur, so it needs to be combined with the second method.

9.1.2 Pilot-based Estimation

It's based on monitoring the channel response using the received pilots' values at the frequency domain. This method alone produces high errors due to the interpolation effect, so it only gives updates to the channel response obtained from the LTF-based estimation.

Both methods need to be implemented, and they're usually very complex due to the implemented division algorithms. The advisor suggested that this block doesn't get implemented to save time, and because it's not needed in short-range communications, which is the case.

The transmission should succeed without this block since the distance between 2 FPGAs is only a few centimeters, but it'll be needed when using the project in Multi-FPGA wireless network.

9.2 Phase Tracking

After the frequency estimation the small mismatch error rotates the constellation with respect to the receiver's demapper; which produces error in the transmitted symbols. Hence, an algorithm should overcome this problem using the estimated channel and the pilots sub-carriers.

9.3 RF Chain

The main objective of the project is to implement the transmitter and receiver on different FPGAs, which is already done, but for verification of the project, the transmitter and receiver need to be connected to RF chains. The USRP board was intended to be used in this role, but since another team was assigned this objective, our progress in this point was linked with theirs. Unfortunately, the team couldn’t reach the goal of connecting the transmitter on FPGA to the transmitting USRP and the receiver on FPGA to the receiving USRP.

An approximate model was used for verification though. The transmitter’s output was added to an approximate channel model, which is AWGN channel with SNR = 5 dB, as suggested by the communications professors. The channel’s output was passed to the receiver and the bits were decoded with 0 error.

9.4 Latency

One critical issue in the Wireless FPGA idea is the latency introduced by the standard. Simply, in the wired communication case, the partitioned design’s different parts would communicate with each other on different FPGAs through wires. These wires connect the FPGAs without any communication protocol, so they don’t add any cycle, so their propagation delay adds significant delay to the maximum delay of the implemented design, which reduces the maximum clock frequency of the design.

If wireless interconnections are used, no delay would be added to the signals, instead, an initial latency is introduced between the transmitting and receiving FPGAs. Simply, the transmitter and receiver’s digital chains each contain serial blocks that add significant latency to the signals due to the registers in the chains. This issue would immerge if any communication protocol is implemented, so it only limits the partitioning tool, as this kind of communication is only half-duplex, so the partitioning tool should partition the designs such that the internal signals between their parts are in 1 direction only.

9.5 Different Modulation Coding Schemes

Since the implemented standard is 802.11n, SISO system, the standard's available modes of operation are from MCS = 0 to MCS = 7. All MCSs are implemented in the transmitter and work successfully in the behavioral simulation. Concerning the receiver, the implemented Viterbi Decoder only works with R = 1/2 encoding, and due to the limited time we didn't have time to implement the other encoding rates, so since the highest rate MCS with R = 1/2 is MCS = 3, the implemented transmitter and receiver work at MCS = 3, and they both work successfully on the FPGAs.

The next step that needs to be done is to implement all other MCSs of the transmitter on FPGA, and implement the Viterbi decoder to work behaviorally with all encoding rates to enable the implementation of all MCSs of the receiver.

9.6 Standard Verification

The method used in the verification of the transmitter's operations was passing its output to the receiver and making sure the received bits have 0 errors without channel effects. Though it was useful, it's not the best approach since some blocks can be implemented in the transmitter and receiver in a wrong way that doesn't appear from this method, so the best method for verification is to use the MATLAB function *wlanWaveformGenerator* that generates Wi-Fi packets. The function was used in the verification of the preamble, but it should have been used in the verification of the transmitter's different operations.

10. Conclusion

This project is considered the first step on the way of wireless communication among FPGAs which will facilitate the prototyping applications. The main target of this project is to find an alternative method of wired connection among FPGAs due to the issues discussed in chapter 1, so an accurate literature review was done on different wireless protocols. Due to some requirements like high data rate and reasonable operating carrier frequency, Wi-Fi was selected as the standard to connect the FPGAs. Wi-Fi has many versions but the IEEE 802.11n version was selected because of the restriction of bandwidth added by the available RF chain "USRP". After that, a MATLAB model and RTL implementation were done for all blocks of the transmitter, receiver, and synchronization blocks, then the transmitter and receiver were implemented on FPGA ZYNQ ZC702 Evaluation Board, and the results of utilization were shown in chapter 8.

This project is now ready to be connected with the USRP to form a real connection between the transmitter and the receiver to achieve the criteria set by the IEEE 802.11n and satisfies the requirements of the project.

11. References

[1] Li, Wen-Kai, et al. "A 2× 2 802.11 ac WiFi transceiver supporting per channel 160MHz operation in 28nm CMOS." *Radio Frequency Integrated Circuits Symposium (RFIC), 2017 IEEE*. IEEE, 2017.

[2] Chen, Tsung-Ming, et al. "7.1 An 802.11 ac dual-band reconfigurable transceiver supporting up to four VHT80 spatial streams with 116fs rms-jitter frequency synthesizer and integrated LNA/PA delivering 256QAM 19dBm per stream achieving 1.733 Gb/s PHY rate." *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*. IEEE, 2017.

[3] Peng, Cheng, Wu Bin, and Hei Yong. "Design and implementation of IEEE 802.11 ac MAC controller in 65 nm CMOS processProject supported by the National Great Specific Project of China (No. 2012ZX03004004_001)." *Journal of Semiconductors* 37.2 (2016): 025002. 116-121.

[4] Guntur, Surendra, et al. "Design of a multi GBPS single carrier digital baseband for 60GHz applications and its FPGA implementation." *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*. IEEE, 2013.

[5] Choi, Sung-Woo, Il-Gyu Kim, and Jae-Min Ahn. "Architecture of baseband modem board for IEEE802. 11ad technology." ICT Convergence (ICTC), 2013 International Conference on. IEEE, 2013.

[6] Choi, Chang-Soon, et al. "60-GHz OFDM systems for multi-gigabit wireless LAN applications." Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE. IEEE, 2010.

[7] Manavi, Farzad. Implementation of OFDM modem for the physical layer of IEEE 802.11 a standard based on Xilinx Virtex-II FPGA. Diss. Concordia University, 2004

[8] Haene, Simon, David Perels, and Andreas Burg. "A real-time 4-stream MIMO-OFDM transceiver: system design, FPGA implementation, and characterization." *IEEE Journal on Selected Areas in Communications* 26.6 (2008).

[9] https://www.ettus.com/content/files/b200-b210_spec_sheet.pdf

[10] 802.11-2016 IEEE Standard for Information Technology Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[11] James W. Cooley, Peter A. W. Lewis, and Peter W. Welch, "Historical notes on the fast Fourier transform," Proc. IEEE, vol. 55 (no. 10), p. 1675–1677 (1967).

[12] C. Yu, M.-H. Yen, P.-A. Hsiung, and S.-J. Chen, "A low-power 64-point pipeline FFT/IFFT processor for OFDM applications," IEEE Transactions on Consumer Electronics, vol. 57, no. 1, pp. 40–40, 2011.

[13] Shah, D. C., B. U. Rindhe, and S. K. Narayankhedkar. "Effects of cyclic prefix on OFDM system." Proceedings of the International Conference and Workshop on Emerging Trends in Technology. ACM, 2010.

[14] Forney, G. David. "The viterbi algorithm." *Proceedings of the IEEE* 61.3 (1973): 268-278.

[15] Heiskala, Juha, and John Terry Ph D. *OFDM wireless LANs: A theoretical and practical guide*. Sams, 2001.